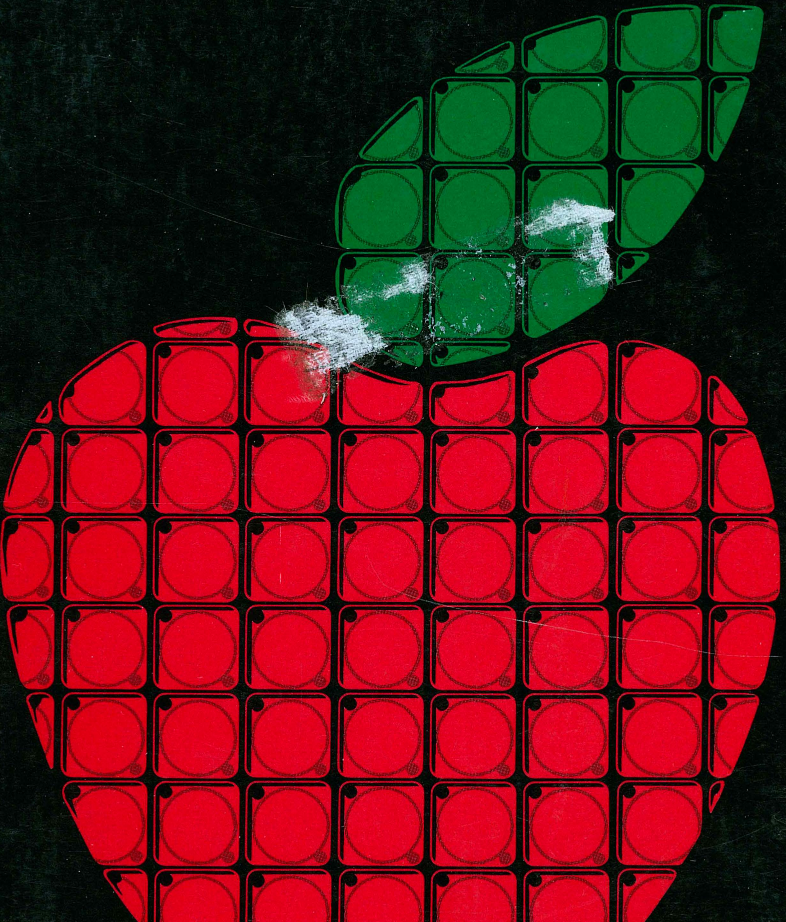
MICROBOOK: DATABASE MANAGEMENT
FOR THE APPLE II®



by Ted Lewis

MICROBOOK:

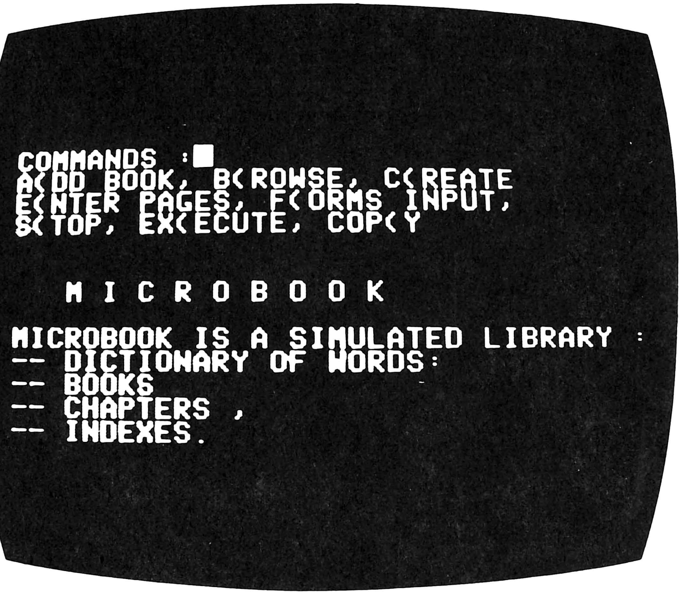
Database Management for the Apple® II Computer

203-5480
B. Dalton
PRICE
$1095
500
3D

MICROBOOK:

Database Management for the Apple® II Computer

T. G. Lewis



```
COMMANDS :█
A(DD BOOK, B(ROWSE, C(REATE
E(NTER PAGES, F(ORMS INPUT,
S(TOP, EX(ECUTE, COP(Y

   M I C R O B O O K

MICROBOOK IS A SIMULATED LIBRARY :
-- DICTIONARY OF WORDS:
-- BOOKS
-- CHAPTERS ,
-- INDEXES.
```

dilithium Press
Beaverton, Oregon

©Copyright, dilithium Press, 1982

All rights reserved. No part of this book may be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system without permission in writing from the publisher, with the following two exceptions: any material may be copied or transcribed for the non-profit use of the purchaser; and material (not to exceed 300 words and one figure) may be quoted in published reviews of this book.

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Lewis, T.G. (Theodore Gayle), 1941-
MICROBOOK: database management for the Apple II computer.

Includes index.
1. Apple II (Computer)—Programming. 2. MICROBOOK (Computer system) 3. Data base management. 4. PASCAL (Computer program language) I. Title.
QA76.8.A662L48 001.64′2 82-7457
ISBN 0-88056-072-X (pbk.) AACR2

Cover: Anton C. Kimball

Printed in the United States of America

dilithium Press
11000 S.W. 11th Street
Beaverton, Oregon 97005

Apple® is a registered trademark of Apple Computer.

Preface

MICROBOOK started as a research project in database design and implementation at Oregon State University. The first version of this experimental system grew from lectures given by the author into a crude database system running on the Apple II computer. During the summer of 1981, the author redesigned and re-implemented the experimental version to make it easier to use and faster to run.

My purpose in publishing this book and the associated programs is to make the *MICROBOOK* system widely available at a low cost to microcomputer users. In this spirit, the program diskettes are also available at minimal cost from the publisher. These programs are complete and ready to run. They were tested over a one-year period before they were made available to the public. However, should you find new bugs, please report them so they can be eliminated.

I would like to thank my typist, Donna Lee Norvell-Race for preparing this manuscript, and my publisher, Merl Miller for his support in making *MICROBOOK* generally available.

T. G. Lewis
May 1982

AN IMPORTANT NOTE

The publisher and author have made every effort to assure that the computer programs and programming information in this publication are accurate and complete. However, this publication is prepared for general readership, and neither the publisher nor the authors have any knowledge about or ability to control any third party's use of the programs and programming information. There is no warranty or representation by either the publisher or the authors that the programs or programming information in this book will enable the reader or user to achieve any particular result.

The programs in this book are available on diskette from dilithium Software, a division of dilithium Press. To order the software please use the order form in the back of this book.

Contents

1. WHAT IS A DATABASE SYSTEM? 1
   1.1 The Parts of a DBMS 1
   1.2 Physical Models 1
   1.3 Logical Models 3
   1.4 Query Languages 5

2. BRIEF OVERVIEW OF MICROBOOK 9
   2.1 What is MICROBOOK? 9
   2.2 How Does MICROBOOK Work? 9
   2.3 How is MICROBOOK Used? 12
   2.4 What Can MICROBOOK Do? 13

3. INTRODUCTION TO MICROBOOK 15
   3.1 Hardware and Software Requirements 15
   3.2 MICROBOOK System Disks 15
   3.3 Starting the MICROBOOK System 18
   3.4 MICROBOOK Command Summary 19

4. C(REATEing A DATABASE LIBRARY 23
   4.1 The Purpose of C(REATE 23
   4.2 An Illustration of C(REATE 23
   4.3 Limitation of MICROBOOK 27

5. A(DDing A BOOK TO THE LIBRARY 29
   5.1 Book Names 29
   5.2 Password Protection 29
   5.3 Example 30

6. F(ORMS INPUTing TO A CHAPTER 33
  6.1 The Purpose of F(ORM 33
  6.2 Rules of Forms 33
  6.3 An Example 34
  6.4 How F(ORM Works 40

7. E(NTER PAGES INTO A CHAPTER 45
  7.1 The Purpose of E(NTER 45
  7.2 Example 45

8. B(ROWSEing THE LIBRARY 51
  8.1 The Purpose of B(ROWSE 51
  8.2 The Browse Function Commands 51
  8.3 Exception Conditions 53
  8.4 An Example 53

9. EX(ECUTEing THE QUERY PROCESSOR 65
  9.1 The Purpose of QUERY 65
  9.2 How to Use QUERY 65
  9.3 The QUERY Language 69
  9.4 Structure of the QUERY Language 70
  9.5 Statement Summary 75
  9.6 Illustrations 76

10. COP(Ying THE LIBRARY TO ANOTHER DISK 79
  10.1 The Purpose of COP(Y 79
  10.2 How to Use COP(Y 79

11. MICROBOOK EXTENSIONS 85
  11.1 The Purpose of MICROBOOK Extensions 85
  11.2 How to Use Unit MICROBOOK 85
  11.3 Summary of MICROBOOK Routines 86
  11.4 An Example 89

12. AN EXAMPLE: ACCOUNTS RECEIVABLE 91
  12.1 Setting Up the Forms 91
  12.2 QUERY Processing 97

13. PROGRAM LISTING FOR MICROBOOK 103
  13.1 Overview 103
  13.2 Data Structure 104
  13.3 Modify/Convert 106
  13.4 Listing 109

14. PROGRAM LISTING FOR UNIT 235
14.1 Overview 235
14.2 Listing 237

15. PROGRAM LISTING FOR QUERY 283
15.1 Overview 283
15.2 Listing 285

INDEX 305

1

What Is a Database System?

1.1 THE PARTS OF A DBMS

A DBMS (database management system) is a collection of programs which store, retrieve, and process data stored in files. More importantly, a DBMS encompasses a *unified, integrated model* of file storage and its corresponding retrieval in order to reduce the labor associated with processing large amounts of data.

This definition of a DBMS may seem a bit abstract, so suppose we look at each part of a DBMS before explaining the details. A DBMS typically includes programs for the following:

- An integrated, unified collection of files. The integrated and unified collection of files is often called the *physical model* of the database.
- An integrated, unified *view* of the physical database. The user's *view* is often called a schema or, as we will call it, a *logical model* of the database. The logical model is how users see the database.
- Query language for processing inquiries from users. These inquiries typically consist of information retrieval or searches for information via a key or collection of keys.
- Input/Output control. Typical DBMS's include input and output control for obvious purposes of data entry and report writing.

1.2 PHYSICAL MODELS

The underlying physical model of a DBMS depends highly on the logical level model. However, nearly all DBMS's must organize file structures in a manner which leads to (1) high speed retrieval, (2) keyed access on one or more keys, and (3) proper file maintenance.

A common technique for high speed retrieval is called *hashing* or key-to-address transformation. Hashing is used to calculate the location of a piece of information in the database given its key. For example, if a mailing list consisting of name, address, and zipcode were accessed by the name (as a key), we would need to hash the name (alphabetic string) into the corresponding disk file location (number) before we could retrieve the address and zipcode information.

Unfortunately, hashing does not typically provide for rapid access to files which contain multiple-keyed information, nor does hashing facilitate good file maintenance practices. The high speed retrieval of hashing can be used in only the most special cases.

The most flexible and powerful physical level mode of a database is the B-tree structure. A *B-tree index* file is a tree-structured file containing the values of all keys to the information stored in a data file. Each access to the information stored in the database must first access a B-tree index file to find the location of the information. In fact, an access to the B-tree index file is required for every key used to make a retrieval.

Figure 1.1 shows the B-tree index files as triangular objects while the file containing a record of information is shown as a rectangle. Notice that one B-tree index file is used for each key.

A B-tree index file structure also has the advantage that all entries (keys) are stored sorted in alphabetical and/or numerical order. This means we can easily retrieve the records in Figure 1.1 by name, zipcode, or age *in order from smallest to largest.* Hence, JONES, LEWIS, and SMITH are accessed in alphabetical order resulting in records #2, #0, and #1 being retrieved.

MICROBOOK employs a B-tree physical model so that every key in the database is stored in a B-tree index file as well as some master file. In fact, a MICROBOOK database may consist of dozens of B-tree index files, one for each access key needed by the user.

We will not delve into the inner workings of B-tree file structures. For a more detailed explanation the reader is re-

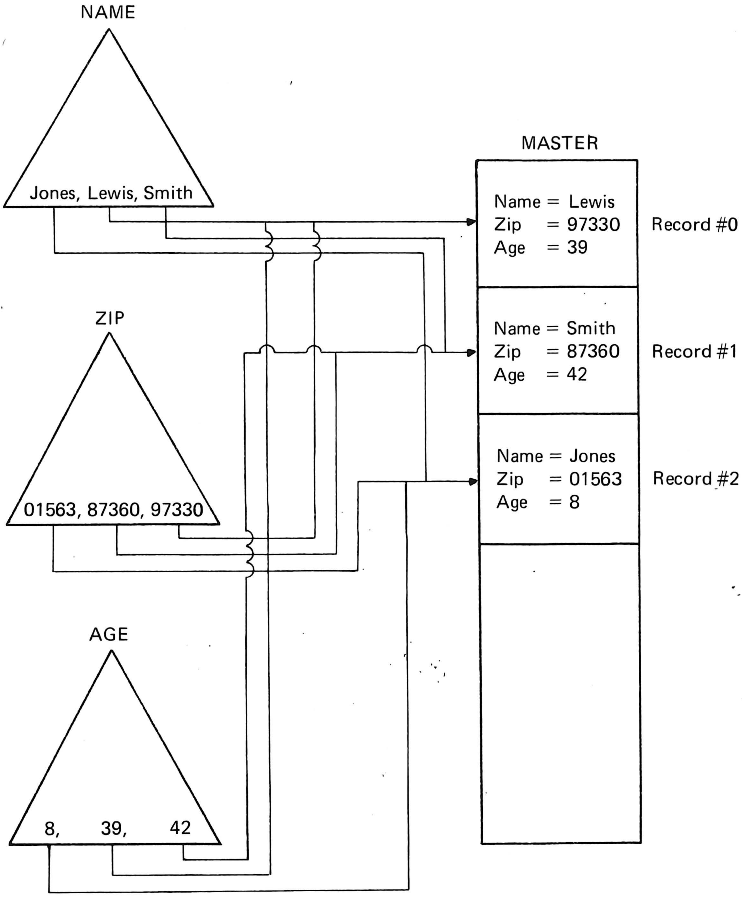

NAME
Jones, Lewis, Smith
ZIP
01563, 87360, 97330
AGE
8, 39, 42
MASTER
Name = Lewis
Zip = 97330
Age = 39
Record #0
Name = Smith
Zip = 87360
Age = 42
Record #1
Name = Jones
Zip = 01563
Age = 8
Record #2

Figure 1.1 Indexed access to multiple-key file

ferred to the author's book, *Pascal Programming For the Apple,* Reston Books, 1981. However, programs for building and accessing keys stored in a B-tree index file are given in this book. See the program listings.

1.3 LOGICAL MODELS

The logical model or logical schema of a database system is the user-perceived structure of the DBMS. It is independent of the physical database and in a way it covers up the physical structure so the user need not think in terms of hashing or B-trees.

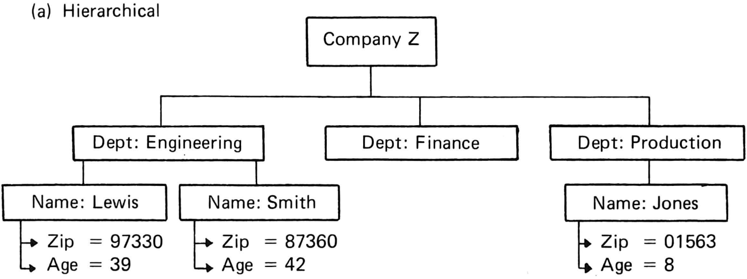
There are three general approaches to designing a logical database:

HIERARCHICAL: Information is viewed as a hierarchical structure with a "top" and "bottom." An example of this view is shown in Figure 1.2(a).

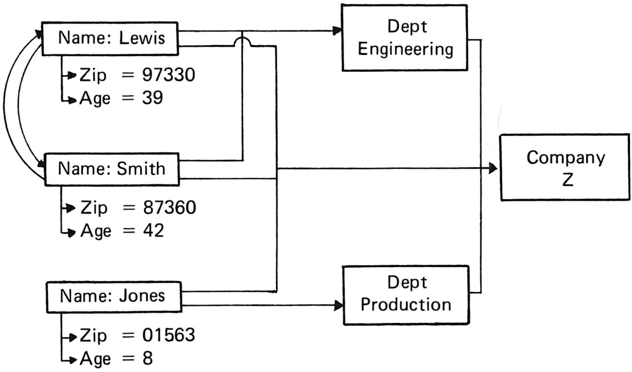
NETWORK: Information is viewed as a network of interacting structures with no apparent structure. This is illustrated by the network of Figure 1.2(b).

RELATIONAL: Information is viewed as a collection of tables called *relations.* This is shown by example in Figure 1.2(c).

(a) Hierarchical


Company Z
Dept: Engineering
Dept: Finance
Dept: Production
Name: Lewis
Zip = 97330
Age = 39
Name: Smith
Zip = 87360
Age = 42
Name: Jones
Zip = 01563
Age = 8

(b) Network


Name: Lewis
Zip = 97330
Age = 39
Name: Smith
Zip = 87360
Age = 42
Name: Jones
Zip = 01563
Age = 8
Dept Engineering
Company Z
Dept Production

(c) Relational

Company Z

Name:	Dept	Age	Zip
Lewis	Engr.	39	97330
Smith	Engr.	42	87360
Jones	Prod.	8	01563

Figure 1.2 Three logical models

In the hierarchical view every employee is part of a department. Therefore, to access information about an employee, we must search down through the levels of the hierarchy. Furthermore, to retrieve other employees in the same department, we can quickly locate them by searching down to the DEPT level. However, to retrieve all employees of the same age, for example, we may need to search the entire hierarchical structure.

The network view uses links (pointers, as shown by arrows) to associate properties of employees with departments, each other, and so forth. The network view is quite flexible due to the arbitrary linking, but it is not easily implemented at the physical level. Therefore, the network view is rarely employed in practical DBMS's.

The relational view offers the most elegant and simple solution to a DBMS logical organization. Each relation is very similar to a table as shown in Figure 1.2(c). The labeled columns are called *domains* and the rows are called *tuples.* We could retrieve all employees in the same department by searching the domain equal to "ENGR", in Figure 1.2(c). Furthermore, we could retrieve all information associated with an employee by reading a tuple from the relation, e.g., LEWIS, ENGR, 39, 97330. Hence, a tuple closely corresponds to the information stored in an ordinary file.

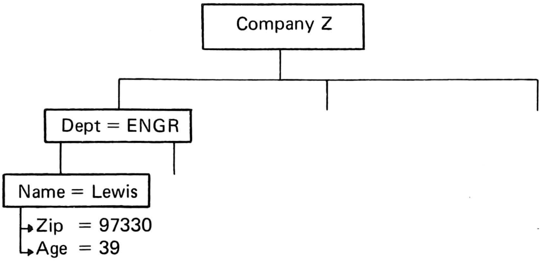
Relational DBMS's are popular because they have such a simple logical structure. However, there are some kinds of processing that a relational DBMS cannot easily perform. For this reason we have adopted a combination of both relational and hierarchical structures for use in MICROBOOK.

1.4 QUERY LANGUAGES

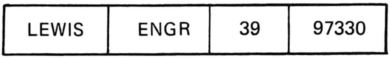
The form of a DBMS query language will conform to the form of its logical view. Thus, a hierarchical query will use tree-searching to locate and output information stored in the hierarchy. For example, the System 2000 query language uses commands to establish a trace through the hierarchy as shown in Figure 1.3(a) and as governed by the query.

The QUALIFY command derives a trace or path from top-to-bottom as shown in Figure 1.3(a). The PRINT statement causes the path to be used to find NAME, AGE, and ZIP, and then print them.

(a) System 2000 Query Qualification (trace).


Company Z
Dept = ENGR
Name = Lewis
Zip = 97330
Age = 39

(b) Relational Query


LEWIS
ENGR
39
97330

(c) Microbook Query (Turn to correct page in the book)

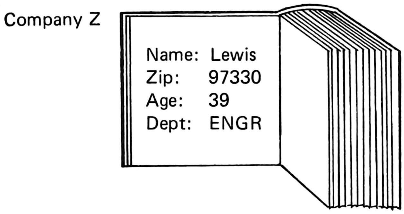

Company Z
Name: Lewis
Zip: 97330
Age: 39
Dept: ENGR

Figure 1.3 Query processing

This approach is used in MICROBOOK to browse the database. However, the simplicity and convenience of a relational query language is needed in more sophisticated processing. Therefore, MICROBOOK employs a very high-level query language much like the query language of INGRES. Here is an example of an INGRES query:

```
RANGE OF T IS COMPANY
RETRIEVE (T. ALL) WHERE NAME="LEWIS"
```

The INGRES query above uses keyword ALL to abbreviate the names of all domains in the tuple corresponding to "LEWIS". This has a corresponding keyword in the Query Language Processor of MICROBOOK. In MICROBOOK we simply enter the statement

```
FOR *NAME="LEWIS" to "LEWIS" DO;
?;
END ;
```

This statement causes all records with NAME equal to "LEWIS" to be displayed on the screen of the Apple computer. For more details on the Query Language Processor turn to the chapter which describes this system. Also, the Pascal program for implementing QUERY can be found at the end of this book.

In the following chapters we will discuss and illustrate by example the MICROBOOK database system implemented on the Apple II microcomputer. All programs (source and object) can be obtained by ordering them from the publisher. See the ordering card attached to this book.

2

Brief Overview of MICROBOOK

2.1 WHAT IS MICROBOOK?

MICROBOOK is a data management system and program development tool designed to turn your Apple II microcomputer into a filing cabinet, information gathering/retrieval system, and data processing engine. It is a collection of programs written in Pascal which turns the computer into a simulated library.

2.2 HOW DOES MICROBOOK WORK?

MICROBOOK simulates a library by maintaining a data dictionary, books, chapters, pages, and indexes to pages all on diskette. You need only think about your problem in terms of facts and data stored in books. The software of MICROBOOK does all bookkeeping and information retrieval. Also, a special processing program called QUERY is included in the MICROBOOK package to allow you to retrieve information, change database facts, and output results. QUERY is an English-like language that is easy to use and requires only a minimum knowledge of computers.

To use MICROBOOK you simply create a book by giving it a name, say PAYROLL, and then create one chapter for every form used by PAYROLL. The details of how to do this are explained in the chapter on the ADD function.

Every form is given a chapter heading, say EMPLOYEE, and this heading is used to reference *pages* containing actual values stored in the chapter. In addition, you must specify the name of every piece of data to be stored in a page. These names are called *data words,* and their values are called *data values.*

Suppose we design a form for EMPLOYEE as follows:

Name: ____

Soc. Sec. # ____________________________________

Pay Rate $ ____________________

Hrs. Worked ___________________

This form is stored in book PAYROLL and chapter EMPLOYEE. Furthermore, we must give unique names to each blank in the form. For example, we might call the blank for holding the employee's name, NAME. The social security number is given the data word SOC, the pay rate data word is called PAYR, and the data word for holding the hours worked is called HRS.

Now, the EMPLOYEE form looks like the following:

Name: ______________ NAME ______________________

Soc. Sec. # ___________ SOC ____________________

Pay Rate $ _________ PAYR _________

Hrs. Worked ________ HRS _________

This defines the EMPLOYEE form and all the data words used in the chapter called EMPLOYEE.

We can think of every page of EMPLOYEE as a form like the one shown above plus data values put in place of data words. Thus, the following two pages of information are stored in book PAYROLL and chapter EMPLOYEE:

Page 0. Name: Smith
Soc. Sec. #455-52-2341
Pay Rate $7.85
Hrs. Worked: 40

Page 1. Name: Jones
Soc. Sec. #528-30-2991
Pay Rate $8.35
Hrs. Worked: 48

Hence, a chapter defines a form and gives names to every value stored in MICROBOOK. Later on when we want to retrieve a page of information we need only use the data word corresponding to the data value we want to retrieve.

MICROBOOK also keeps an index for every book and chapter. This index contains the data values of every *key word* in the library. A key word is a data word that has been marked by the user for access later on. Key words are designated with an asterisk prefix. For example, we can make NAME and PAYR key words by calling them *NAME and *PAYR. The EMPLOYEE chapter would look as follows in this case:

Name: *NAME

Soc. Sec. # SOC

Pay Rate $ *PAYR

Hrs. Worked: HRS

Now, we can enter pages of data values just as before and index them automatically as we do so. Later, we can retrieve any page we want using either *NAME or *PAYR. MICROBOOK automatically provides all access information needed to retrieve all pages with matching data values for either *NAME or *PAYR. For details on how this is done, see the chapter on the BROWSE function.

A typical MICROBOOK database might look like the structure shown in Figure 2.1. The hierarchical organization of the library in Figure 2.1 suggests that a MICROBOOK database is a hierarchical database. This is true, but a MICROBOOK database is also *relational* because we can retrieve any page from any book/chapter by simply requesting the system to retrieve all data values corresponding to a key word. This is the job of the QUERY processor. See the chapter on the QUERY Language Processor for details on how this is done.

Therefore, MICROBOOK is a data management system that allows you to define your database as a hierarchical library to fit the mold of a business, and then to retrieve information from the library in a relational manner. This combination of hierarchical and relational structure is the essence of power and flexibility in MICROBOOK.

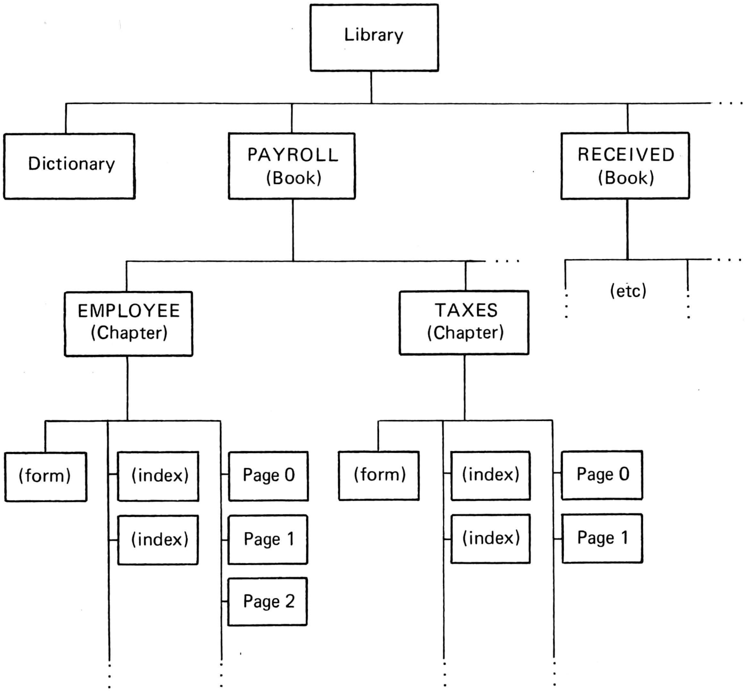

Library
Dictionary
PAYROLL (Book)
RECEIVED (Book)
(etc)
EMPLOYEE (Chapter)
TAXES (Chapter)
(form)
(index)
(index)
Page 0
Page 1
Page 2
(form)
(index)
(index)
Page 0
Page 1

Figure 2.1 The structure of MICROBOOK – An Example

2.3 HOW IS MICROBOOK USED?

When using MICROBOOK you must specify an application in terms of input and output forms. The input and output forms are stored in a book chapter as discussed above. Then when the data ENTRY function of MICROBOOK is used, new pages of data values can be added to any book/chapter in the library. These pages are kept in the system as shown in Figure 2.1. In addition, an index is automatically constructed as the data values are entered. See the chapter that describes how to use the ENTRY function.

Once data values are entered into a book/chapter, they can be retrieved automatically using the BROWSE function built into MICROBOOK. The chapter that describes the BROWSE function gives details on its use.

In addition, book/chapter pages can be modified, displayed, and updated automatically by any query program you write. This program consists of English-like commands that are recognized and processed by the QUERY processor. To understand how this system works, see the chapter which describes the QUERY Language Processor.

Finally, you can maintain an audit trail and purge unneeded books from a library by copying portions to another disk. In addition, the library may be increased in size (more pages) by making a backup copy onto another disk. This is described in greater detail in the chapter on the COPY function.

2.4 WHAT CAN MICROBOOK DO?

MICROBOOK can be used for almost any application where storage and retrieval of information is important. Some examples are given here, but the uses of MICROBOOK are only limited by your imagination. It is a tool to be used in new and creative ways. It is a tool for increasing productivity.

A *real estate* database can be implemented with the aid of MICROBOOK by creating a book called HOMES and two chapters: 4SALE and SOLD. The 4SALE chapter might list all homes for sale indexed by price, school district, and number of bedrooms. The SOLD chapter might list the homes that have been sold in the past six months, so that comparables can be retrieved based on price, size (floor space), etc. Additional information can be kept also, but these two chapters can be used to rapidly retrieve information for buyer, broker, and appraiser alike.

An *accounting system* might be implemented in MICROBOOK. For example, a dentist or doctor can create a book called BILLING which contains three chapters, MASTER, TRANS, and STMTS. The pages in MASTER contain pertinent patient information like name, address, and outstanding balance. The TRANS chapter holds pages containing charges and/or payments. The STMTS chapter is a form for printing bills to be sent at the end of each month.

In this example, we might use a query program that processes all pages of TRANS at the end of the month and updates the outstanding balance for patients that have a transaction pending. This can be done in a few short commands using QUERY.

A *graphics system* might employ MICROBOOK to store data for graphical display. A book called PLOTS might have a chapter called GRAPH (line drawings), one called PIE (for pie charts), and one called BAR (for bar charts). The ENTRY function of MICROBOOK is used to enter data points, and then a user supplied program is written to display any plot upon demand.

This example would be implemented by extending MICROBOOK using the routines stored in SYSTEM LIBRARY. For details on this process see the chapter on MICROBOOK Extensions.

Other applications of MICROBOOK are listed below in order to give you some ideas:

record collection library
building contractor estimation
musical scores
family trees
sports statistics
tax database
foreign language translations
grocery lists and menus
stock market histories
educational testing and quizzes
dating service lists

This is by no means an exhaustive list. The other applications of MICROBOOK are up to you!

3

Introduction to MICROBOOK

3.1 HARDWARE AND SOFTWARE REQUIREMENTS

MICROBOOK is a data management system and program development tool designed to accelerate the implementation of software on the Apple II microcomputer. It runs on a one or two disk drive 48KB Apple II with a language card for supporting the Pascal high level language. In addition, it is advisable to use 80-column, 24-line video output like the output provided by the Videx card in order to take maximum advantage of the forms-handling capability of MICROBOOK. An optional printer can be used to obtain hard copy output. The illustrations in this book will assume a 40-column screen, no printer, and two disk drives (#1 and #2).

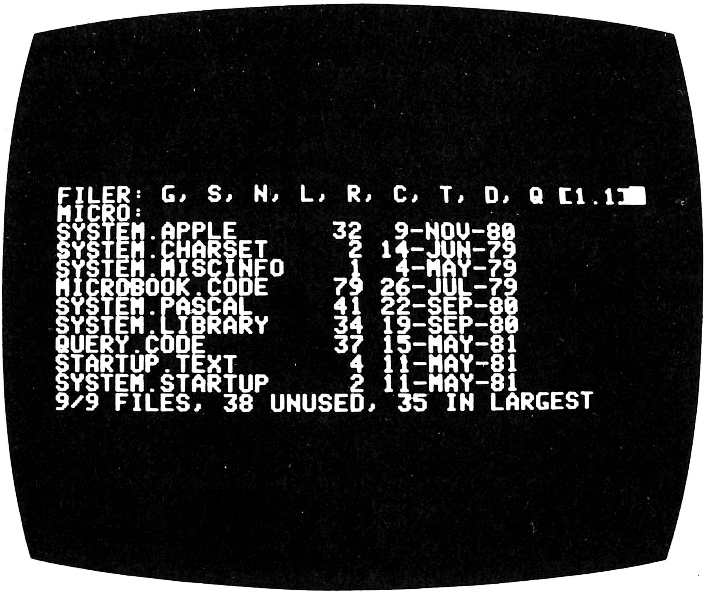
Illustrations were selected to demonstrate how to use MICROBOOK. Internal details of the system have been omitted in favor of an application-oriented explanation.

All photographs were taken directly from the screen using actual data and practical examples. Explanations have been organized so that a first-time user can learn to use MICROBOOK reading this book from start-to-finish.

3.2 MICROBOOK SYSTEM DISKS

The MICROBOOK software package contains one disk called MICRO. This disk should be used to build two disks as follows:

MICRO: this disk should contain the programs for performing all functions of MICROBOOK as described in this book. Figure 3.1 shows the titles of the pro-



```
FILER: G, S, N, L, R, C, T, D, Q [1.1]
MICRO:
SYSTEM.APPLE         32  9-NOV-80
SYSTEM.CHARSET        2 14-JUN-79
SYSTEM.MISCINFO       1  4-MAY-79
MICROBOOK.CODE       79 26-JUL-79
SYSTEM.PASCAL        41 22-SEP-80
SYSTEM.LIBRARY       34 19-SEP-80
QUERY.CODE           37 15-MAY-81
STARTUP.TEXT          4 11-MAY-81
SYSTEM.STARTUP        2 11-MAY-81
9/9 FILES, 38 UNUSED, 35 IN LARGEST
```

Figure 3.1 Contents of MICRO:

grams that should be put on the directory of this disk. This disk is placed in drive #1 when it is used. The most important files of interest in Figure 3.1 are MICROBOOK.CODE, QUERY.CODE, STARTUP.TEXT, and SYSTEM.STARTUP.

MICROBOOK.Code

This file contains the executable (P-code) program called MICROBOOK. This program is automatically executed each time you start or initialize your Apple II. We will explain the various functions of MICROBOOK throughout the rest of this book. But for now recall that MICROBOOK allows you to create a library of books, chapters, and pages. Each book corresponds to a *database* containing information supplied by you. Each chapter corresponds to a *form* for entering or retrieving the database information. Finally, each chapter contains pages corresponding to the database information, itself. We will see how these components of MICROBOOK are used in subsequent illustrations.

Query.Code

This file contains the executable (P-code) program called QUERY. The QUERY program is actually an English-like language processor that allows you to process the information stored in any MICROBOOK database. The QUERY language processor is a very high level language for instructing MICROBOOK to retrieve, store, modify and in general process books, chapters, and pages. For example, QUERY can be used to post accounts receivable transactions, payroll data, personnel information, and so forth. An entire section is devoted to QUERY processing (see the chapter on the QUERY Language Processor).

Startup.Text
System.Startup

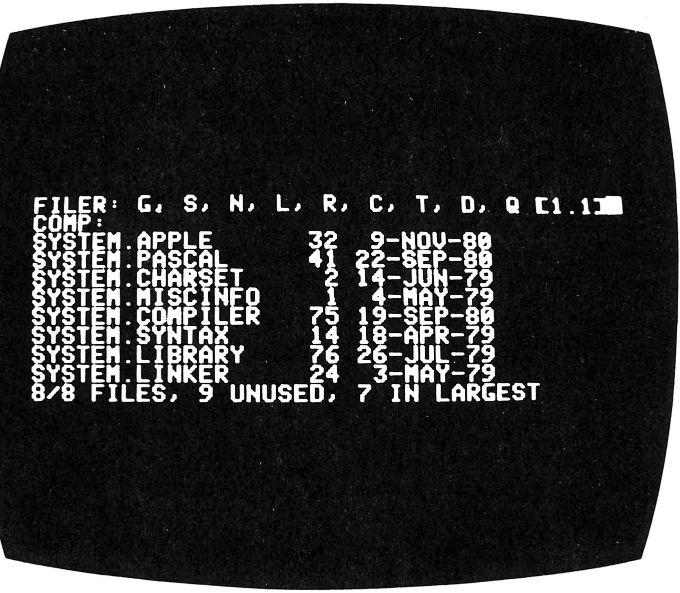
These files contain a simple Pascal source program called STARTUP.TEXT and its executable equivalent called SYSTEM.STARTUP for automatically executing MICROBOOK each time the MICRO: disk is initialized or the Apple II turned on with MICRO: in disk drive #1. If you do *not* wish for MICROBOOK to be automatically executed, then these two files should be removed from MICRO:.

COMP: This disk is used to extend the capabilities of MICROBOOK by adding new processing programs to the basic system. This will be useful only to programmers that have a detailed knowledge of Pascal programming.

This disk should contain software provided by Apple Computer Company (see Figure 3.2). The only file of interest here is the one titled SYSTEM.LIBRARY.

System.Library

The file contains all library routines provided by Apple Computer Company plus new routines in a special unit called MICROBOOK. The MICROBOOK unit contains pre-compiled routines which perform information retrieval, input, output, searching, numerical conversions, and so forth. These routines are extremely useful to users familiar with Pascal because they can be used in any Pascal program to rapidly



```
FILER: G, S, N, L, R, C, T, D, Q [1.1]
COMP:
SYSTEM.APPLE         32  9-NOV-80
SYSTEM.PASCAL        41 22-SEP-80
SYSTEM.CHARSET        2 14-JUN-79
SYSTEM.MISCINFO       1  4-MAY-79
SYSTEM.COMPILER      75 19-SEP-80
SYSTEM.SYNTAX        14 18-APR-79
SYSTEM.LIBRARY       76 26-JUL-79
SYSTEM.LINKER        24  3-MAY-79
8/8 FILES, 9 UNUSED, 7 IN LARGEST
```

Figure 3.2 Contents of COMP:

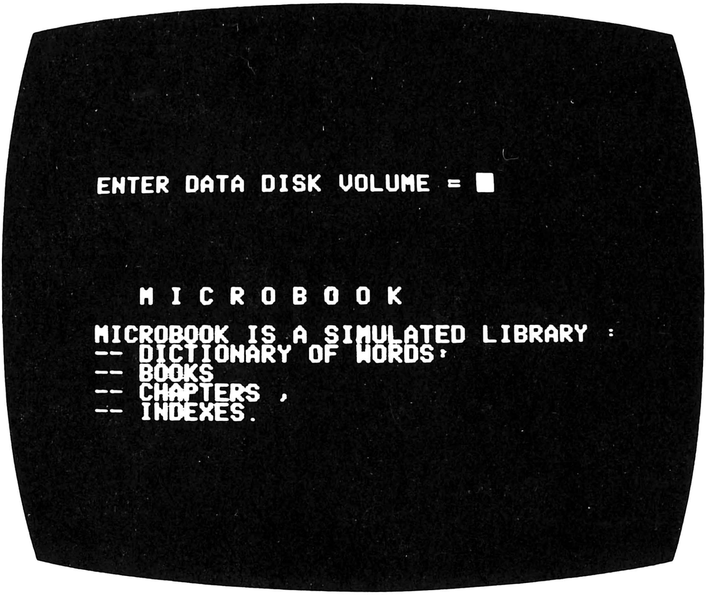
access and process MICROBOOK databases. For more information on this feature of MICROBOOK, see the chapter on MICROBOOK Extensions.

3.3 STARTING THE MICROBOOK SYSTEM

To start using MICROBOOK you must first build a disk containing the files shown in Figure 3.1. Use the FILER provided by the Pascal System to copy (T=Transfer) files from MICRO: to a new disk called COMP: as shown in Figure 3.2. Next, compress MICRO: using the FILER crunch command (K=crunch). Finally, copy the system routines listed in Figure 3.1 onto the MICRO: disk.

Now you can insert MICRO: into drive #1 and any blank diskette into drive #2. We will use a diskette called DBASE: in drive #2 throughout the examples. Either RESET, INITIALIZE, or POWER-ON the machine and sit back and wait approximately 30 seconds for MICROBOOK to boot up and display a sign on the message screen.

Figure 3.3 shows what happens when MICROBOOK is ready to use. The screen will be filled with the MICROBOOK



```
ENTER DATA DISK VOLUME = ■


    M I C R O B O O K

MICROBOOK IS A SIMULATED LIBRARY :
-- DICTIONARY OF WORDS,
-- BOOKS
-- CHAPTERS ,
-- INDEXES.
```

Figure 3.3 Startup MICROBOOK

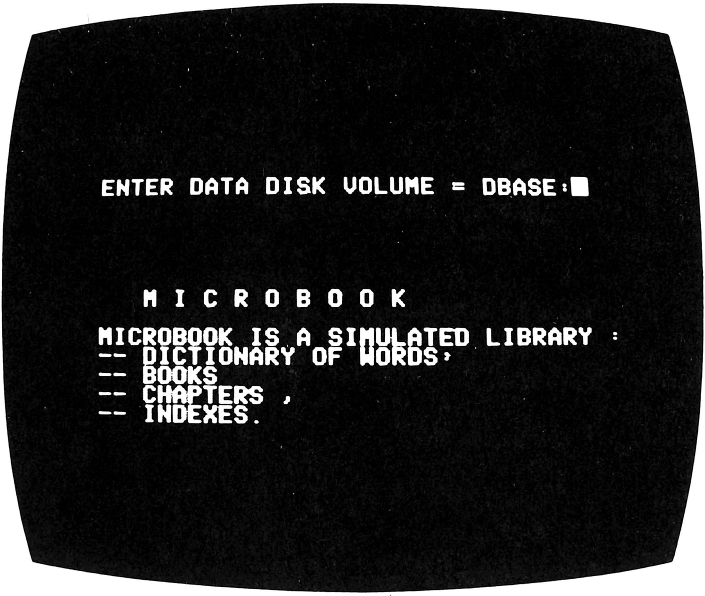
banner and then the computer will wait for you to enter the name of the disk volume to be used to hold the MICROBOOK library, books, etc.

Figure 3.4 shows the result of entering DBASE: for the name of the database disk. MICROBOOK will remember to include the trailing colon at the end of the volume name if you forget. Also, the name DBASE: is terminated by a RETURN key.

Figure 3.5 shows what happens when the disk volume name DBASE: is entered followed by pressing the RETURN key. This is called the *command level* of MICROBOOK because all functions of MICROBOOK are accessible from this level.

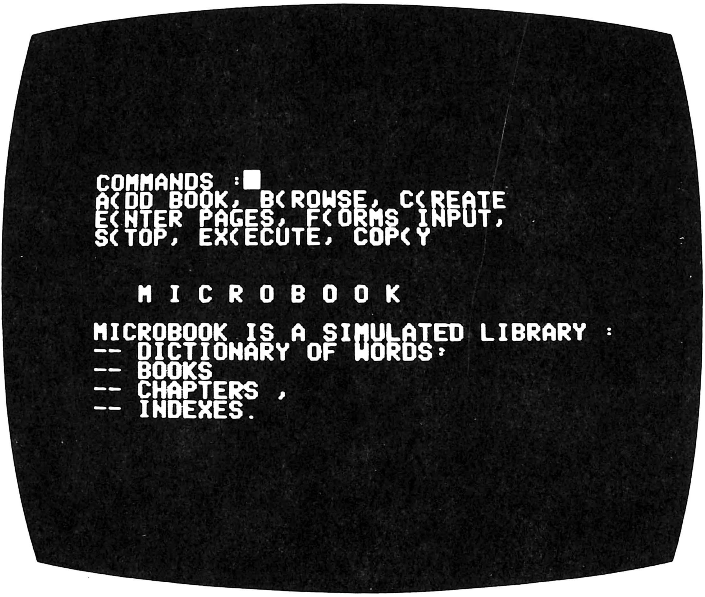
3.4 MICROBOOK COMMAND SUMMARY

Notice in the command level of Figure 3.5 that each command is shown as a single letter followed by "(" to designate a MICROBOOK command. This type of "menu driven" user interface notation will be used throughout MICROBOOK to prompt you for commands. Hence, the eight commands shown in Figure 3.5 are activated by pressing a single letter



```
ENTER DATA DISK VOLUME = DBASE:█



   M I C R O B O O K

MICROBOOK IS A SIMULATED LIBRARY :
-- DICTIONARY OF WORDS,
-- BOOKS
-- CHAPTERS ,
-- INDEXES.
```

Figure 3.4 Enter name of database volume



```
COMMANDS :█
A(DD BOOK, B(ROWSE, C(REATE
E(NTER PAGES, F(ORMS INPUT,
S(TOP, EX(ECUTE, COP(Y

   M I C R O B O O K

MICROBOOK IS A SIMULATED LIBRARY :
-- DICTIONARY OF WORDS,
-- BOOKS
-- CHAPTERS ,
-- INDEXES.
```

Figure 3.5 The command level of MICROBOOK

key (without a RETURN key). That is, A, B, C, E, F, S, X, and P are the only functions that MICROBOOK can perform:

A: A(DD BOOK. This function allows you to add a new book to the database library. Hence, DBASE: will contain another book after this command is executed.

B: B(ROWSE. This function allows you to browse (retrieve) the database by retrieving books, chapters, and pages stored in the library. It also allows you to examine the data dictionary to see what data words have been used by various books and chapters.

C: C(REATE. This function erases any previously established library and initializes the database disk. This should be used when first starting to use MICROBOOK, or when a new database is being created for the first time.

E: E(NTER PAGES. This function is used to enter data values into books and chapters that have been established beforehand. This is used during a data capture phase or whenever you want to add more pages of data to a book/chapter.

F: F(ORMS INPUT. This function is used to define a new chapter, its form, and its length (number of pages). This function also is used to define the data words (key words) corresponding to each form "blank."

S: S(TOP. This returns control back to the Pascal Operating System. To get back from the Pascal Operating System the user must type "I."

P: COP(Y. This is used to make a backup copy of the library. It is also used to extend the size of any book/chapter so that larger books can grow and shorter books can be compressed. Also, COP(Y can be used to remove books that are no longer needed. The original database disk is not altered, but instead a new database disk is created containing only those books and chapters that are still active.

To activate one of the functions described above, simply press the A, B, C, E, F, S, X, or P, key and the corresponding command will be carried out. Then turn to the chapter in this book which describes how to use the corresponding function.

A typical sequence of commands is given below as a guide to the beginner:

C: Create and initialize the database.

A: Add a book and its password.

F: Add a chapter and its password. Also, create a form, data words, and the pages to hold information.

E: Enter values into the pages of a chapter, one page at a time. Continue this until the book is full or until all pages desired have been added.

X: Execute the QUERY processor. This allows you to define several processing steps that can be performed on the books in the library.

P: Copy the database so that a backup copy is retained. This provides an audit trail and protection against loss of valuable information.

After a book and chapter has been defined, you will typically add other books and chapters. Hence, the sequence below may be typical:

A: Add another book.

F: Add another chapter.

F: Add another chapter.

E: Enter data.

B: Browse to see what is in the book.

E: Enter more data.

B: Browse some more.

Now, turn to the appropriate chapter to see how to do each one of the functions discussed above.

4

C(REATEing a Database Library

4.1 THE PURPOSE OF C(REATE

The purpose of the C command is to create a fixed size data dictionary called MICROBOOK. DICT, create a library file called MICROBOOK.LIB, and initialize these files. *The C command causes all existing books to be removed from the library,* so do not execute this command unless you want to start anew!

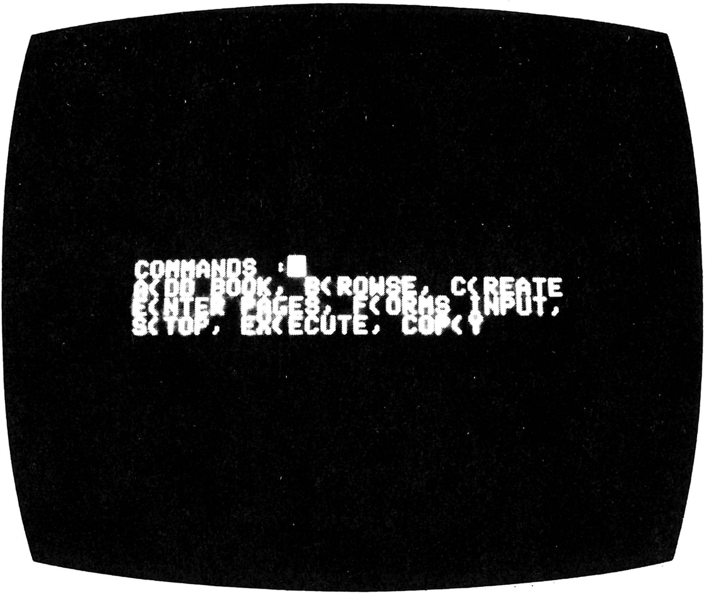
In addition, C displays all system parameters along with their values. Failure to observe these limits will result in either a *value range error,* or else a *string overflow* error during data entry or data processing.

4.2 AN ILLUSTRATION OF C(REATE

Figure 4.1 shows the Command level prior to selecting C for C(REATE. The first input to C(REATE is the maximum number of data dictionary words as shown in Figure 4.2. Twenty-five words are allowed in this example. If the data dictionary becomes full at some later time (because additional forms have been added via books/chapters), then a message is displayed. The only way to extend the data dictionary is to perform a backup using the COP(Y Command.

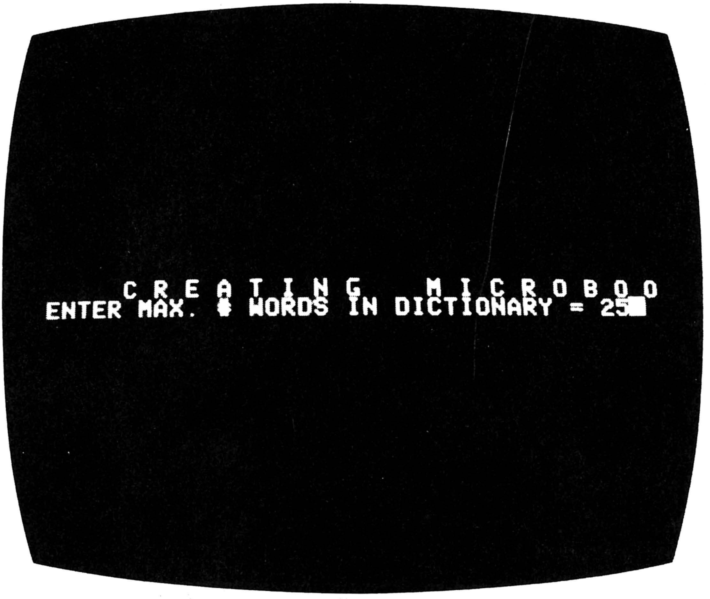
If the data dictionary limit is set too high, the data dictionary will occupy too much disk space. If it is set too low, you will be required to make time-consuming backups. Therefore, it pays to make accurate estimates as to the expected number of data words that you will need.

After entering the size of the data dictionary, MICROBOOK displays the usable region of the screen as shown in Figure 4.3. This is the area of the screen which can be used to build



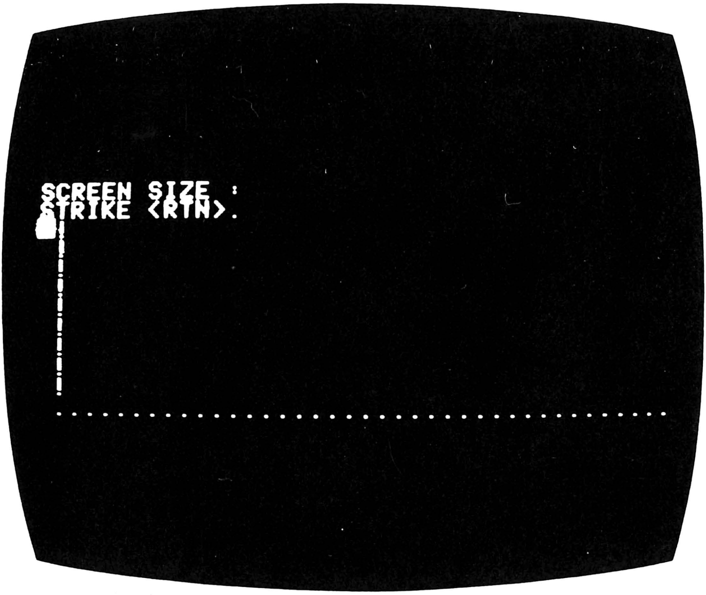
```
COMMANDS :
A(DD BOOK, B(ROWSE, C(REATE
E(NTER PAGES, F(ORMS INPUT,
S(TOP, EX(ECUTE, COP(Y
```

Figure 4.1 Select C(REATE



```
     C R E A T I N G   M I C R O B O O
ENTER MAX. # WORDS IN DICTIONARY = 25
```

Figure 4.2 Enter limit to the number of data words allowed in data dictionary. Maximum size is 25.



```
SCREEN SIZE :
STRIKE <RTN>.
```

Figure 4.3 Screen size is shown

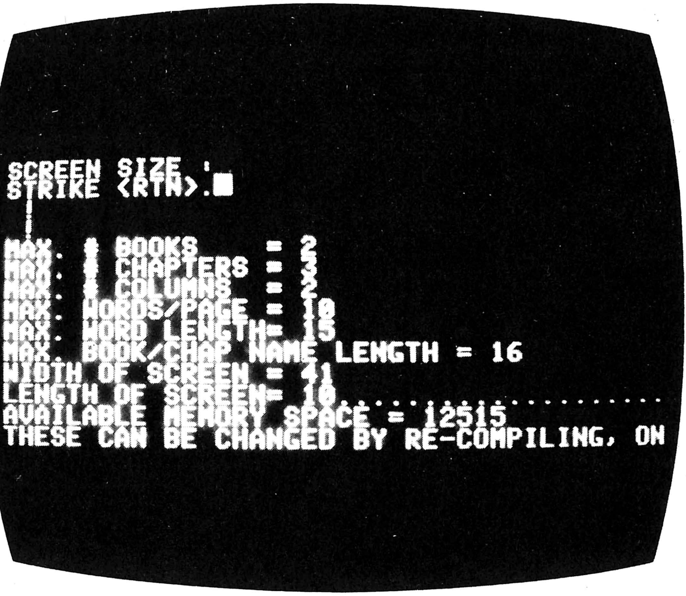
forms and enter data values. MICROBOOK must be recompiled with new values of XLIMIT, YLIMIT, and XORIGIN, YORIGIN to increase or decrease the size of this area. Setting the screen size too high wastes memory (both main memory and disk memory), while setting the screen size too low restricts the capacity of each book/chapter. See the program listing for more details.

Figure 4.4 shows what is displayed after pressing RETURN. The limits of all system parameters are displayed. These values should be kept in mind when using MICROBOOK. They have the following meaning:

MAX.# BOOKS=2. The maximum number of books in MICROBOOK.LIB (library) is set by the program parameter NBOOKS.

MAX.# CHAPTERS=3. The maximum number of chapters per book is set by the program parameter NCHAPS.

MAX.# COLUMNS=2. The maximum number of columns per chapter is set by the program parameter NCOLS. A two-column page is typically used to take advantage of 130 column printers.



```
SCREEN SIZE :
STRIKE <RTN>.■

MAX. # BOOKS        = 2
MAX. # CHAPTERS     = 3
MAX. # COLUMNS      = 2
MAX. WORDS/PAGE     = 10
MAX. WORD LENGTH= 15
MAX. BOOK/CHAP NAME LENGTH = 16
WIDTH OF SCREEN = 41
LENGTH OF SCREEN= 10
AVAILABLE MEMORY SPACE = 12515
THESE CAN BE CHANGED BY RE-COMPILING, ON
```

Figure 4.4 MICROBOOK limitations on sizes

MAX. WORDS/PAGE = 10. The maximum number of *blank fields* or data words per page of a chapter is set by the program parameter NWORDS. This means that no more than 10 values per page can be entered into a chapter.

MAX. WORD LENGTH = 15. The maximum length of a data word or a data value is set by the program parameter MAXWORD. This means that the name of a *blank field* or data word cannot exceed 15 characters. It also restricts the length of an input value to 15 or fewer characters. A numeric input must also be less than or equal to 15 alphanumeric characters long.

MAX. BOOK/CHAP NAME LENGTH = 16. The maximum length of a name used to name books or chapters is set by program parameter MAXCHAP. Names which exceed 16 characters in length will cause a *value range* or *string overflow* error.

WIDTH OF SCREEN = 41. The width of the usable area of the screen is 41 characters. This is set by the program parameter YLIMIT.

LENGTH OF SCREEN = 10. The length (depth) of the screen is 10 lines. This is set by program parameter XLIMIT.

AVAILABLE MEMORY SPACE = 12708. The remaining main memory space is displayed as an indication of the amount of main memory that remains at this point. Increasing any one of the program parameters causes the amount of available space to decrease. If the parameters are set too high, the error message: "STACK OVERFLOW" is displayed and MICROBOOK will not run. This message may occur at any time later on during execution. Therefore, this value must not be allowed to become too small. MICROBOOK has been run successfully with 3,000 words of main memory available, but *beware* because it all depends on the type of processing that is being done.

4.3 LIMITATION OF MICROBOOK

The parameters discussed above are limitations of MICROBOOK. They can be removed by re-compiling both source programs (MICROBOOK and UNIT). However, you are cautioned that increasing these parameters leads to increased memory requirements. Hence, a price must be paid for greater numbers of books, chapters, etc. Practical upper limits of 8 books, 5 chapters, and 2 columns have been achieved in MICROBOOK, but these limits often mean that COP(Y will not execute properly due to a shortage of main memory.

You should also be aware that changes in these parameters render books and chapters incompatible with earlier versions (running with different parameter values). A program called MODIFY is provided to modify these parameters prior to re-compiling, and another program called CONVERT can be used to convert an *old* database into a *new* database. Hence, if a library is created with say NBOOKS = 3 and then it is discovered that NBOOKS = 5 is required, CONVERT can be used to convert the library with 3 books into a library with 5 books. See the program listings for more details.

5

A(DDing a Book to the Library

5.1 BOOK NAMES

The number of characters per book name is specified at the time a library is created. To find out the maximum length of a book name see the chapter on C(REATEing a database library.

A book name can contain letters or digits.

Failure to enter a proper book name will most likely result in a *value range error* or a *string overflow error.* If this happens, simply type a space and the system will be initialized. When initialization is completed, the command level of MICRO-BOOK will appear on the screen and you can make a second attempt to add a book.

5.2 PASSWORD PROTECTION

Every book and chapter in the library *must* have a password. When the name of a book is entered, the prompt

ARE YOU SURE?

will appear. If you answer "Y" (RETURN), the next prompt will be

ENTER BOOK PASSWORD

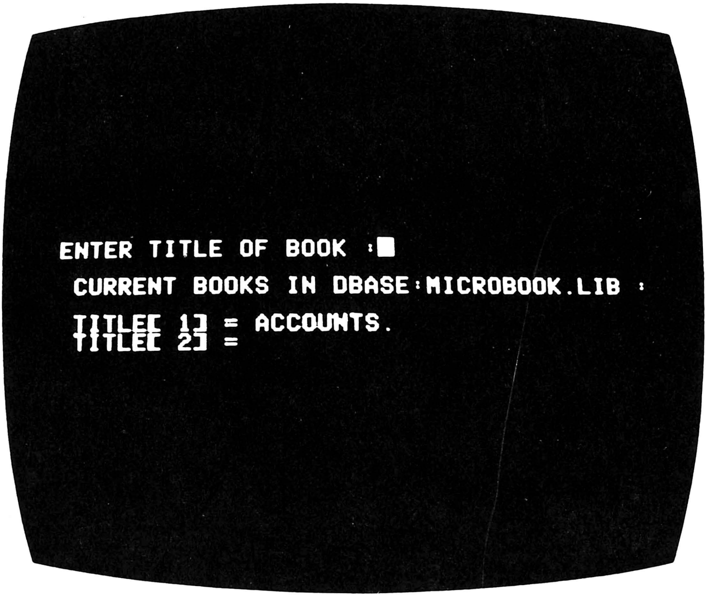
At this time you can type a string of numbers and/or letters as a password. The password *will not appear on the screen* as you type. This password is remembered by MICROBOOK and cannot be displayed or changed.

You must remember all passwords. There is no easy way to find a password once it is forgotten. Therefore, be very careful to use meaningful passwords that can be easily recalled.

Passwords must not exceed the maximum length of book titles as specified when the library is created. Passwords that are too long will cause a *value range error* or a *string overflow error.*

5.3 EXAMPLE

Figures 5.1, 5.2, and 5.3 show some of the input and output required during an A(DD function operation. In Figure 5.1 the system displays the current list of books in the library. Notice that the library is stored on disk volume DBASE: and it is called MICROBOOK.LIB. There is currently one book called ACCOUNTS, and room for one more book in the library.

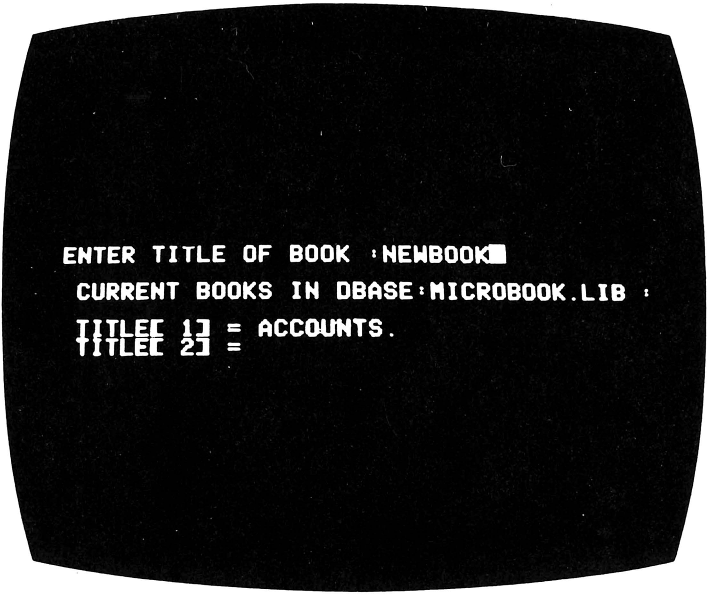


```
ENTER TITLE OF BOOK :■
 CURRENT BOOKS IN DBASE:MICROBOOK.LIB :
 TITLE[ 1] = ACCOUNTS.
 TITLE[ 2] =
```

Figure 5.1 Prompt for name

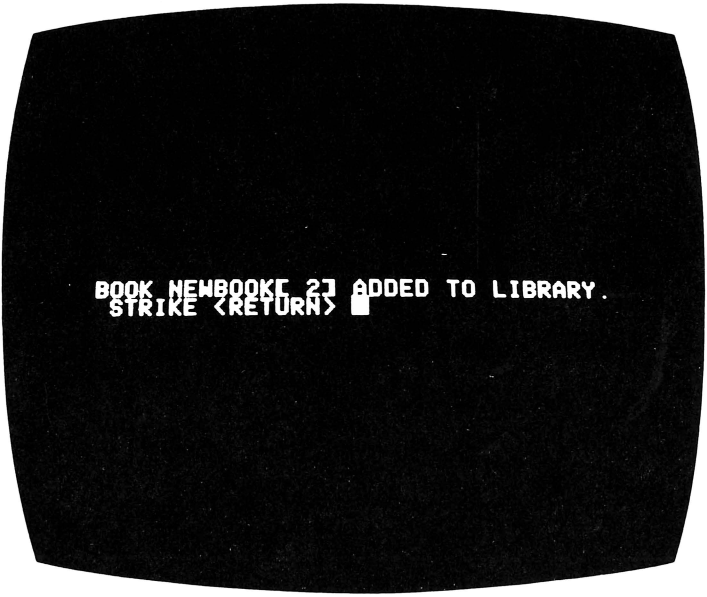
In Figure 5.3 we have entered the name of a new book called NEWBOOK. This name is entered in the library called MICROBOOK.LIB along with ACCOUNTS.

Not shown in the sequence of pictures is a step which gets the password for NEWBOOK from the user. This step is prompted by
ENTER BOOK PASSWORD:
and *will not* display the password as it is entered.



```
ENTER TITLE OF BOOK :NEWBOOK█
 CURRENT BOOKS IN DBASE:MICROBOOK.LIB :
 TITLE[ 1] = ACCOUNTS.
 TITLE[ 2] =
```

Figure 5.2 Enter name



```
BOOK NEWBOOK[ 2] ADDED TO LIBRARY.
 STRIKE <RETURN> █
```

Figure 5.3 Return to command level

Finally, Figure 5.3 shows MICROBOOK: response to a successful A(DD function. The name and location (2) of NEWBOOK are displayed. When you are ready to return to the command level press the RETURN key. MICROBOOK will return to the Command level and wait for another command to be entered.

6

F(ORMS INPUTing to a Chapter

6.1 THE PURPOSE OF F(ORM

Every chapter in every book of the library defines two database entities in MICROBOOK:

(1) Pages containing data values
(2) Forms containing data words

The purpose of the F command is to build a form for a newly created chapter and then define the data words corresponding to the form. Every *blank* entry in a form must have a data word name.

The F command prompts you for information as follows:

(1) Name and password of newly created chapter
(2) Number of columns in the form (and pages)
(3) Name of every data word in the form
(4) Maximum length of the chapter (measured in pages)

These parameters are used to establish a chapter containing room enough for a fixed number of pages, and a form which defines the contents of each page. Note: chapters can be extended by increasing their maximum length (in pages) during a backup COP(Y operation.

6.2 RULES OF FORMS

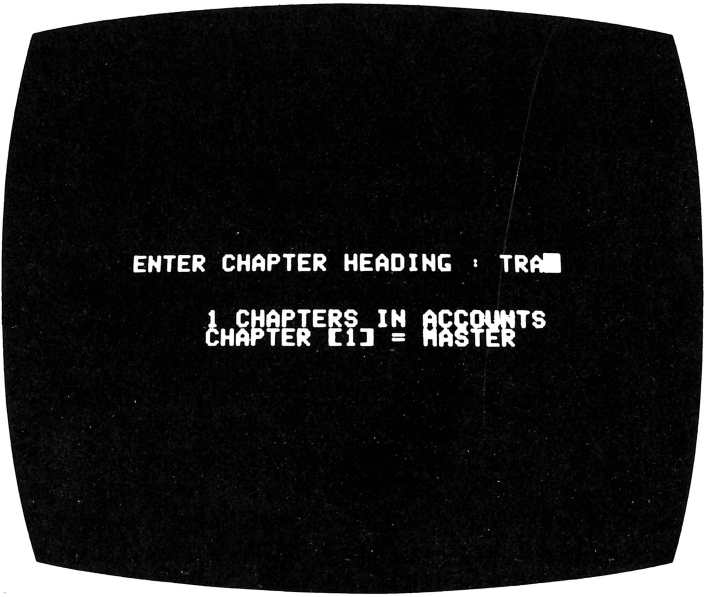
Every input form defines the contents of a chapter. These forms are fixed at the time a chapter is created as shown below. The following rules govern the definition of a form:

(1) A form (page) may consist of either one or two columns (columns are used to put a wide page out to a printer).
(2) Every line of a form must be entered even though it may be blank.

(3) A string of "–" (minus) characters defines a data entry blank. There must not be any other characters imbedded within this string.
(4) A line which begins with a "/" (divide) is suppressed during data entry using the E(NTER function.
(5) Every data word (and corresponding data value) must fit within the space provided by a blank (string of "–"s).
(6) A data word prefixed with an asterisk "*" is automatically marked as a key word. Key words are used to retrieve pages containing certain values corresponding to the key word.

6.3 AN EXAMPLE

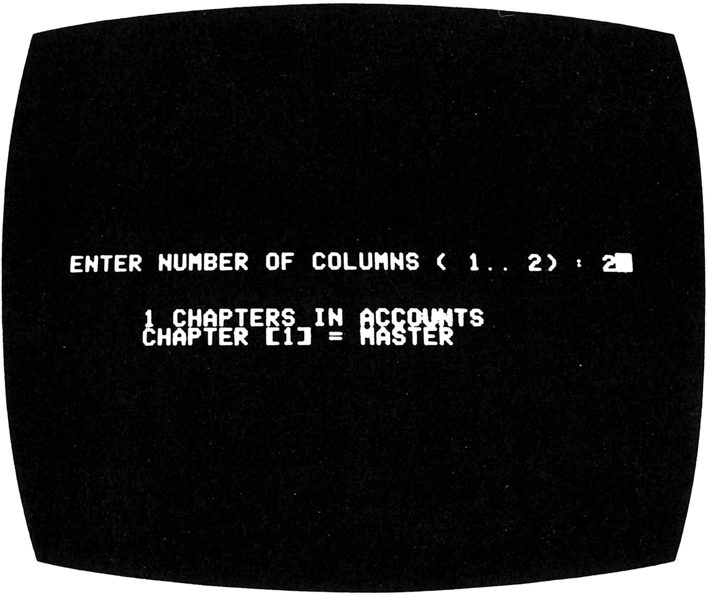
Figure 6.1 shows what happens when the F(ORM INPUT command is executed, the name and password of book ACCOUNTS is properly given, and the name of a new chapter is entered. Suppose we follow what happens as you enter a new chapter called TRANS.



```
ENTER CHAPTER HEADING : TRA█

          1 CHAPTERS IN ACCOUNTS
          CHAPTER [1] = MASTER
```

Figure 6.1 Enter chapter heading

Figure 6.2 shows two columns being selected. Each page of data values will consist of two columns, therefore, the form must consist of two columns. Each column is entered, one at a time.



```
ENTER NUMBER OF COLUMNS ( 1.. 2) : 2

   1 CHAPTERS IN ACCOUNTS
   CHAPTER [1] = MASTER
```

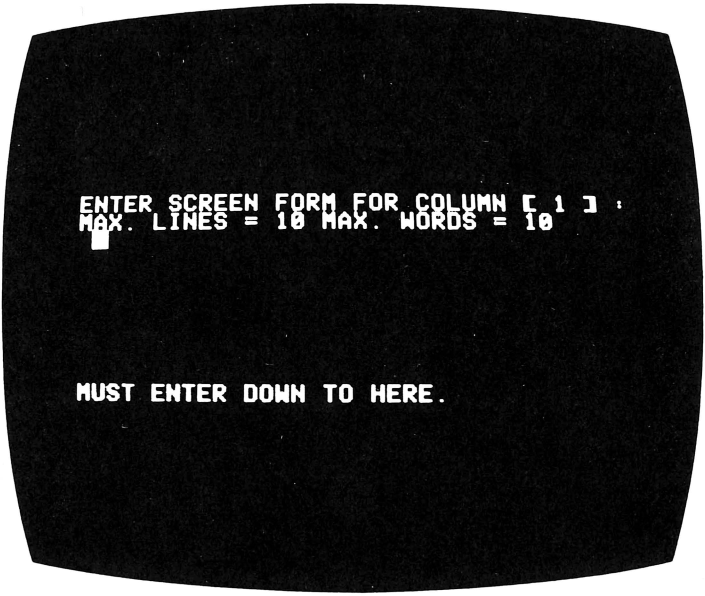
Figure 6.2 How many columns?

Typically, two column forms are used to format output to a printer in MICROBOOK. The two columns are printed side-by-side, thus using the full width of the printer.

Figure 6.3 shows the first step in entering a form. The prompt

MAX. LINES = 10MAX.WORDS = 10

reminds you that a total of 10 lines containing no more than 10 data words (data values) is allowed. You should also be cautious about the width of each blank space in the form. The C(REATE function sets the width (see MAXWORD) to a certain maximum. If any of these constants is exceeded, MICROBOOK will crash.



```
ENTER SCREEN FORM FOR COLUMN [ 1 ] :
MAX. LINES = 10 MAX. WORDS = 10




MUST ENTER DOWN TO HERE.
```

Figure 6.3 Begin defining forms

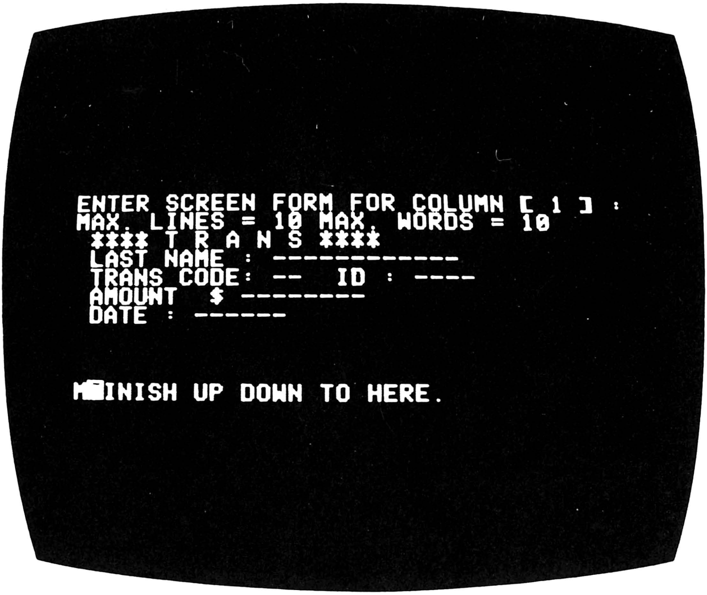
Also, every line of the form space must be specified. If you desire a blank line, them simply press RETURN. Figure 6.4 shows a form of 10 lines; 5 are blank and 5 are filled. Furthermore, the form contains 5 blank spaces as defined by five strings of *minus signs.*

You can edit this form at this point. If you enter a line number between 1 and 10, MICROBOOK will move the cursor to the beginning of the line and allow you to enter a new line. When you desire no additional changes, enter zero.

Figure 6.5 shows the result of entering column two of book ACCOUNTS, chapter TRANS. Column two consists of three non-blank lines and three data entry spaces. Again, corrections can be made a line at a time terminated by a zero.

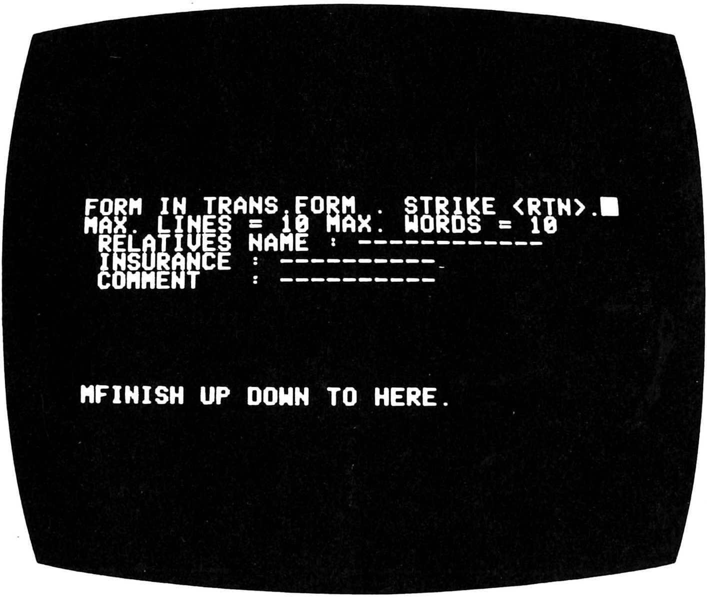
After each column of the form is entered and edited, the system will store the column in a form file. Then MICROBOOK will establish a page file consisting of a certain number of empty pages.

Figure 6.6 illustrates how to specify a chapter consisting of twenty-five empty pages. All twenty-five pages are allocated on disk at this point. If you do not know how many pages will



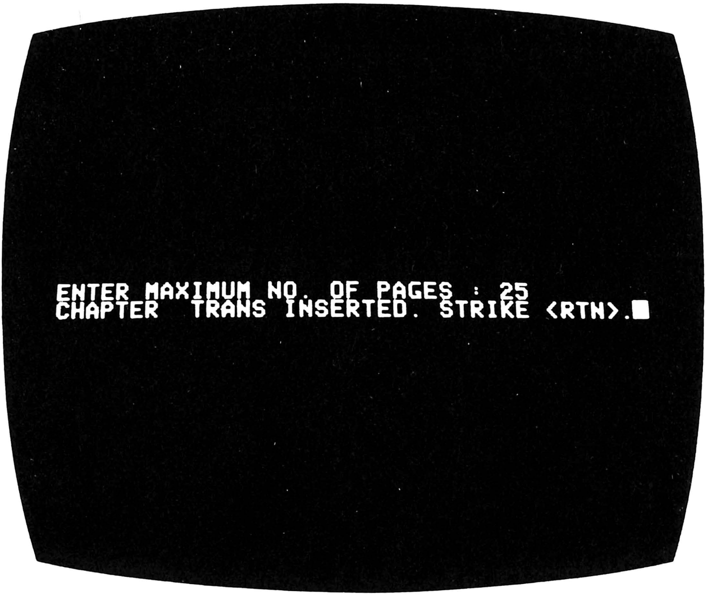
```
ENTER SCREEN FORM FOR COLUMN [ 1 ] :
MAX. LINES = 10 MAX. WORDS = 10
 **** T R A N S ****
 LAST NAME : -------------
 TRANS CODE: --   ID : ----
 AMOUNT  $ ---------
 DATE : ------


M█INISH UP DOWN TO HERE.
```

Figure 6.4 Form is defined, column 1



```
FORM IN TRANS.FORM . STRIKE <RTN>.█
MAX. LINES = 10 MAX. WORDS = 10
 RELATIVES NAME : -------------
 INSURANCE : -----------
 COMMENT   : -----------




MFINISH UP DOWN TO HERE.
```

Figure 6.5 Form is defined, column 2



```
ENTER MAXIMUM NO. OF PAGES : 25
CHAPTER  TRANS INSERTED. STRIKE <RTN>.■
```

Figure 6.6 Maximum number pages is 25

be needed, then a guess will be sufficient. You can increase the length of any chapter by using the COP(Y function to compress and copy books and chapters onto another diskette. See COP(Y command for details.

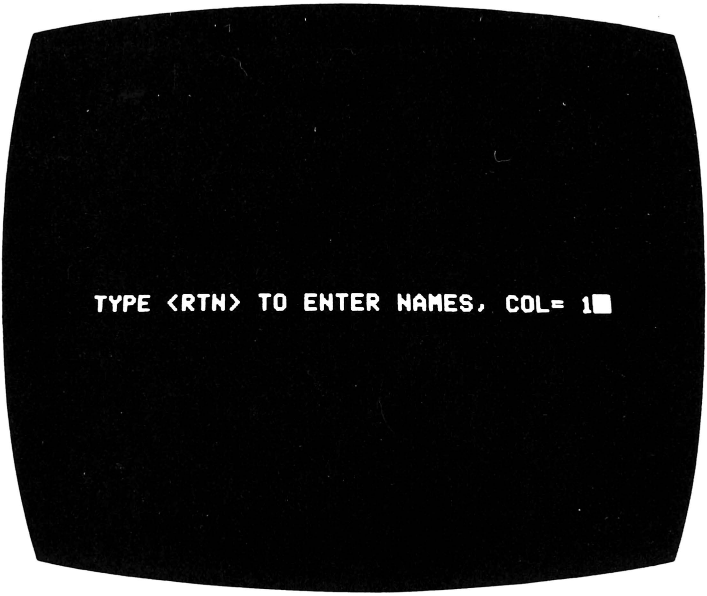
Next, MICROBOOK requests the data names corresponding to each *blank space* in the newly entered form. The sequence of Figures 6.7, 6.8, and 6.9 shows how to enter data names:

*TNAME: This data word refers to the value of the *last name.* Since it is prefixed with an "*", it is also a key word. Access to pages in this chapter can be made via this key word.

TC : The *trans code* value must not exceed two columns in width.

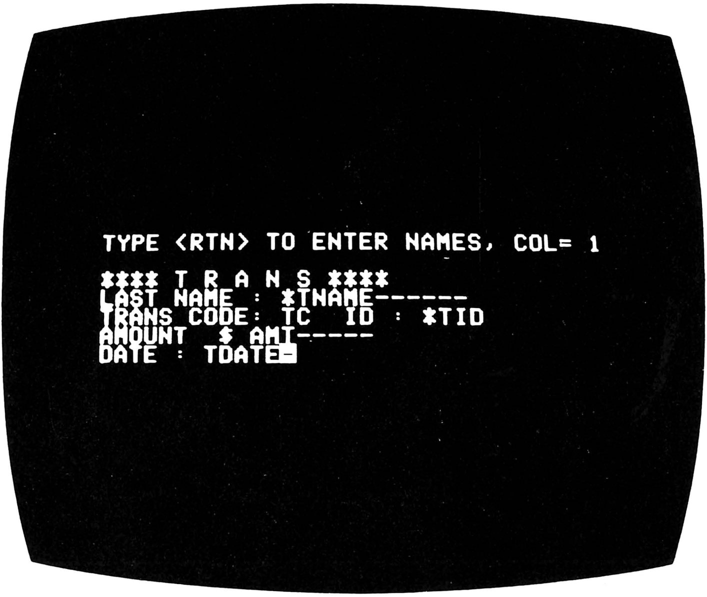
*TID : Again, a data word becomes an access key word whenever it begins with an asterisk. Hence, pages in this chapter can also be accessed by the value of this data word.

AMT : The value of AMT can be a numeric string. NUMERIC strings can be added, subtracted,



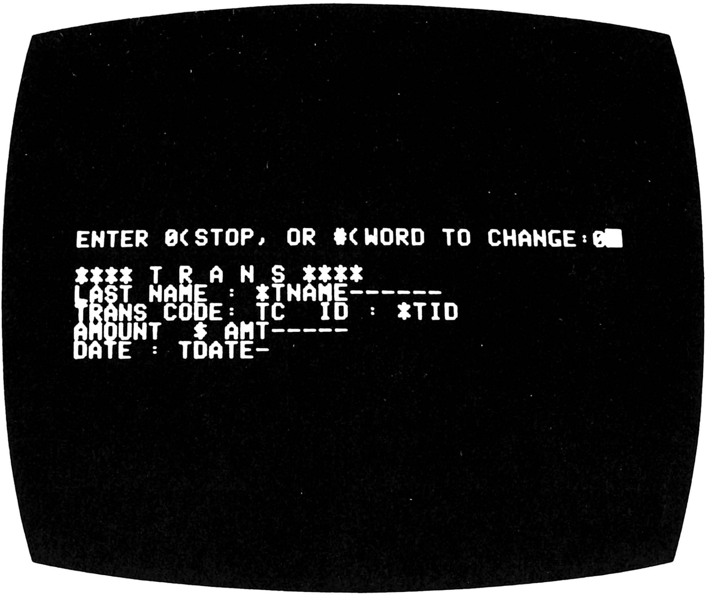
```
TYPE <RTN> TO ENTER NAMES, COL= 1█
```

Figure 6.7 Begin entering data words



```
TYPE <RTN> TO ENTER NAMES, COL= 1

**** T R A N S ****
LAST NAME : *TNAME------
TRANS CODE: TC   ID : *TID
AMOUNT  $ AMT-----
DATE : TDATE-
```

Figure 6.8 Data words for TRANS



```
ENTER 0(STOP, OR *(WORD TO CHANGE:0

**** T R A N S ****
LAST NAME : *TNAME-------
TRANS CODE: TC  ID : *TID
AMOUNT  $ AMT-----
DATE : TDATE-
```

Figure 6.9 No corrections

multiplied, or divided (see the chapter on QUERY).

TDATE : The *blank space* provided for a data word must be contiguous and wide enough to wholly contain the data word.

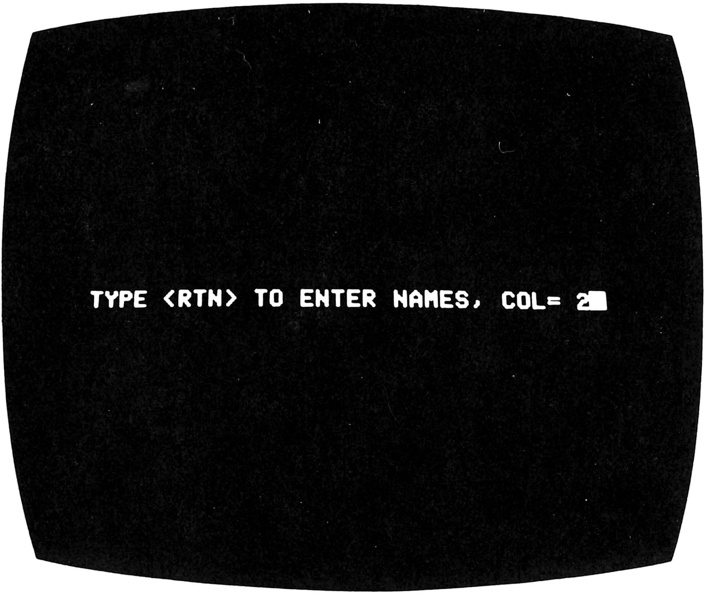
Note: the data words used in a certain form must be unique. Furthermore, all data words used throughout the library of books and chapters must be unique because they are stored (and referenced) in the same data dictionary.

Similarly, Figures 6.10, 6.11, and 6.12 show how to enter data names for column two of chapter TRANS.

6.4 HOW F(ORM WORKS

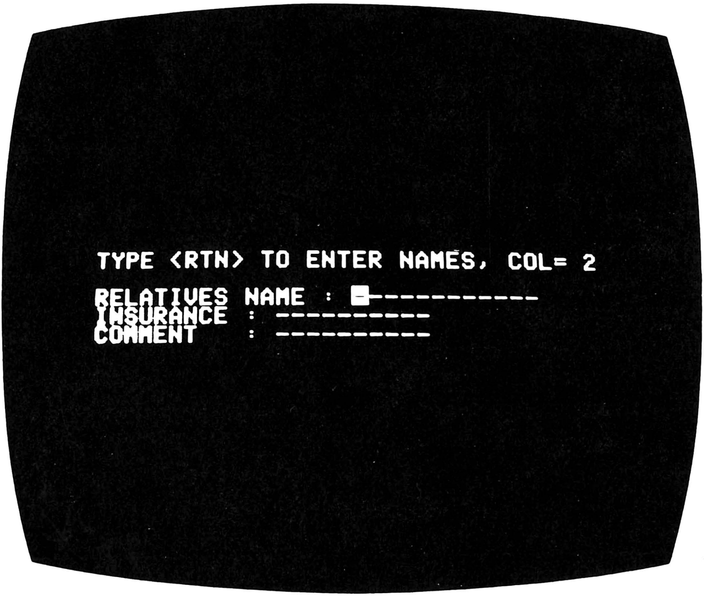
A brief explanation of how the F function processes forms and pages may help you understand MICROBOOK at this point. You may choose to skip this section.

MICROBOOK is designed to minimize the time and effort needed to implement software on the Apple II. A major fraction of time goes into defining forms for capturing input data



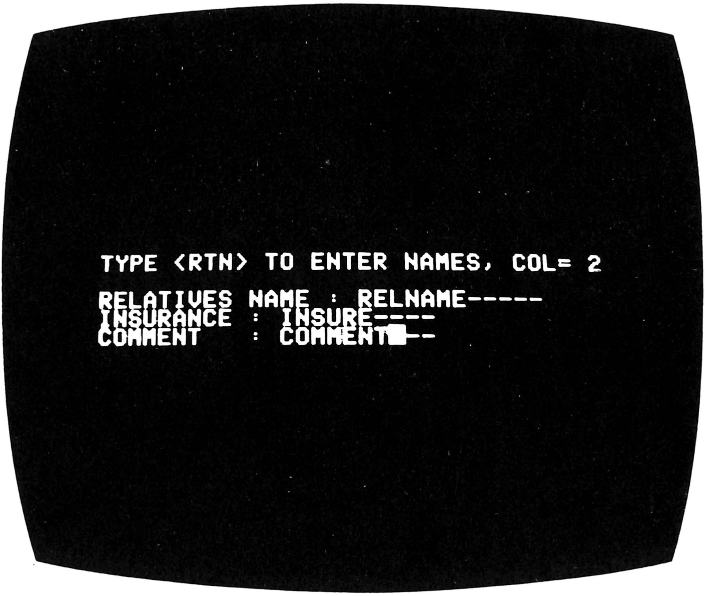
```
TYPE <RTN> TO ENTER NAMES, COL= 2█
```

Figure 6.10 Enter data words, column 2



```
TYPE <RTN> TO ENTER NAMES, COL= 2
RELATIVES NAME : █-----------
INSURANCE : -----------
COMMENT   : -----------
```

Figure 6.11 Form is retrieved, column 2



```
TYPE <RTN> TO ENTER NAMES, COL= 2

RELATIVES NAME : RELNAME------
INSURANCE : INSURE----
COMMENT   : COMMENT█--
```

Figure 6.12 Data words added to data dictionary

and indexing it. Therefore, the purpose of F(ORM is to cut down on this work.

MICROBOOK stores every form in a form file on disk. Whenever data is entered into a chapter page, the form is retrieved and displayed on the screen. Lines beginning with a "/" are *not* displayed because it is assumed they contain values that are to be computed rather than entered.

Secondly, MICROBOOK does away with sorting pages of information by keeping an index file for every key word. The index file is given the same name as the key word (without *). Therefore, each time a data value is entered in a page it is stored in the page, and if it corresponds to a key word, it is also entered in the appropriate index file.

A MICROBOOK library consists of a data dictionary, books and chapters, and one index file for every key word. In addition, every chapter has a form file associated with it which holds the input and display form for the chapter.

You can peruse the database volume directory using the FILER program. The directory will contain files with one of the following suffixes:

*LIB	The library
*DICT	The data dictionary
*BOOK	The table of contents directory for a book
*FORM	The form file of a chapter
*PAGES	The pages of a chapter
*INDEX	The index file for a key word
*AVAIL	Header files for holding parameters

The prefix portion of these files will be recognizable from knowledge of the data words, chapter, and book names.

7

E(NTER PAGES Into a Chapter

7.1 THE PURPOSE OF E(NTER

The E(NTER function is used to enter data values into the pages of a certain chapter. Furthermore, every data value corresponding to a key word is entered into the corresponding index so that the page can be retrieved on the value entered.

The E(NTER function *pre-sorts* the key word values as they are indexed. This makes the data entry step a little slower than desirable but eliminates any need to sort pages of a chapter later on.

7.2 EXAMPLE

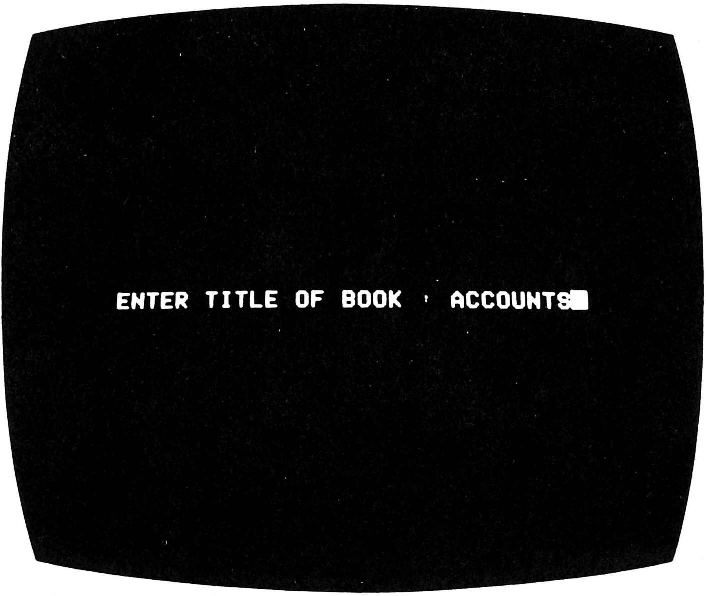
Figures 7.1 and 7.2 show how you must first enter the name and password of book and chapter before entering data into pages.

The prompt line in Figure 7.3 gives you a choice between creating a new file of pages or adding to an already existing file of pages:

N(EW FILE: Erase the previously stored pages and begin entering new pages from page zero.

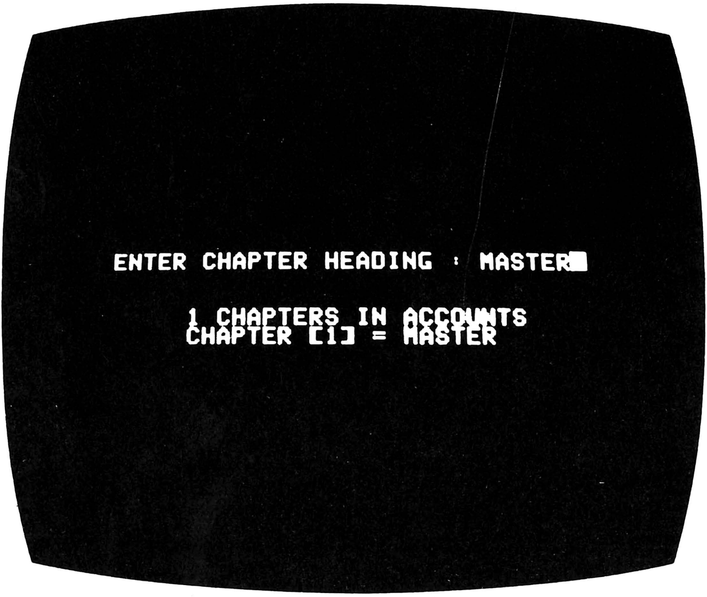
A(DD TO FILE: Find the end of the file of pages and append new pages onto the existing ones.

You are cautioned to *always use N(EW when entering pages the first time.* Failure to do so will cause page zero to be skipped, leading to system problems later on.



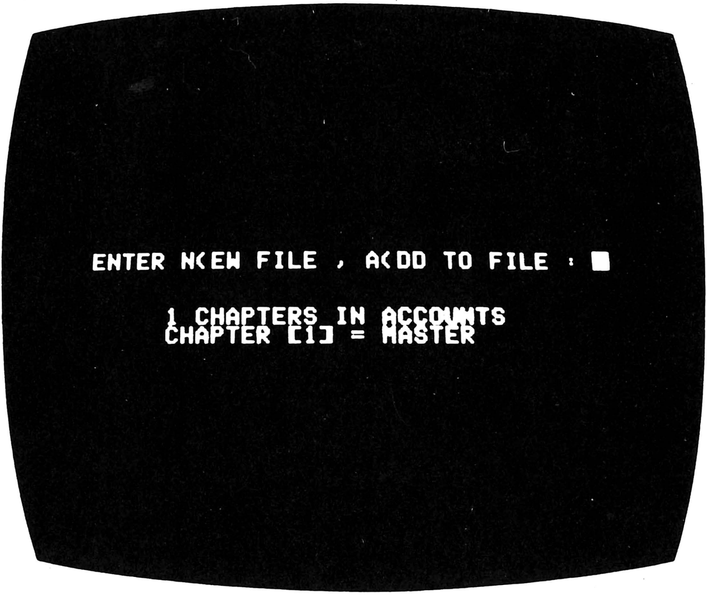
```
ENTER TITLE OF BOOK : ACCOUNTS█
```

Figure 7.1 Enter data values to book ACCOUNTS



```
ENTER CHAPTER HEADING : MASTER█

    1 CHAPTERS IN ACCOUNTS
    CHAPTER [1] = MASTER
```

Figure 7.2 Enter data values to chapter MASTER



```
ENTER N(EW FILE , A(DD TO FILE : ■

1 CHAPTERS IN ACCOUNTS
CHAPTER [1] = MASTER
```

Figure 7.3 Select NEW or ADD to existing chapter?

Figures 7.4 and 7.5 illustrate how to enter data values into page zero. Figure 7.6 shows how to edit the values entered by inputting the number of a *data value* to be changed. For example, LEWIS is data value #1, and CORVALLIS is data value #4, counting from left-to-right, top-to-bottom. Figure 7.7 illustrates how data value #3 is changed.

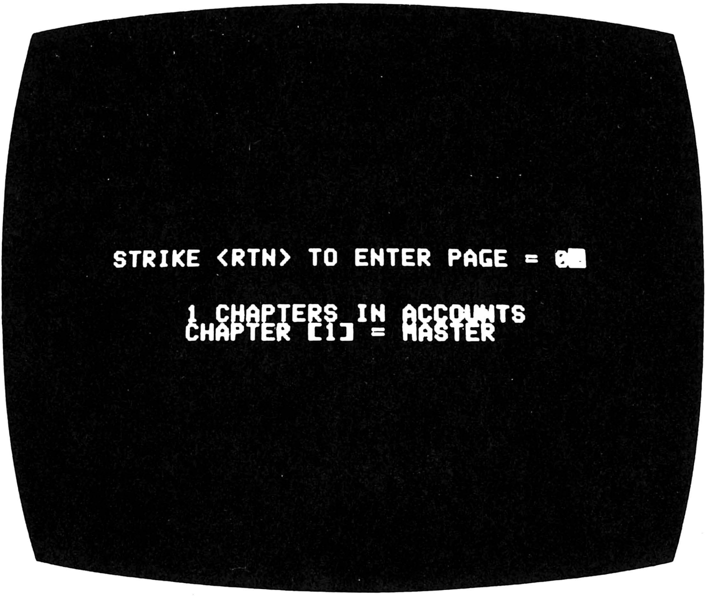
In Figure 7.8 the data values have been stored in page zero and key word values LEWIS, L002, and 97330 have been entered into index files.

Figure 7.9 shows how to stop or add more pages of input:

S(TOP: Return to the Command level of MICROBOOK,

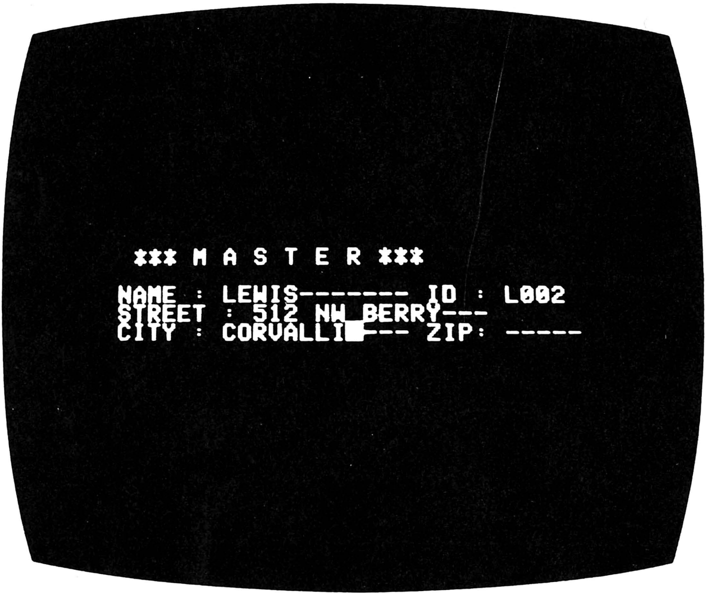
A(DD MORE PAGES: Add another page of data to the chapter. This causes the next page in sequence to be appended to the page file.

The indexing operations will be displayed at the top of the screen after each data value is inserted into the appropriate index. This causes a small (annoying) delay in the data entry step, but recall that it avoids sorting. Hence, the delay is repaid in greater dividends when database processing is done.



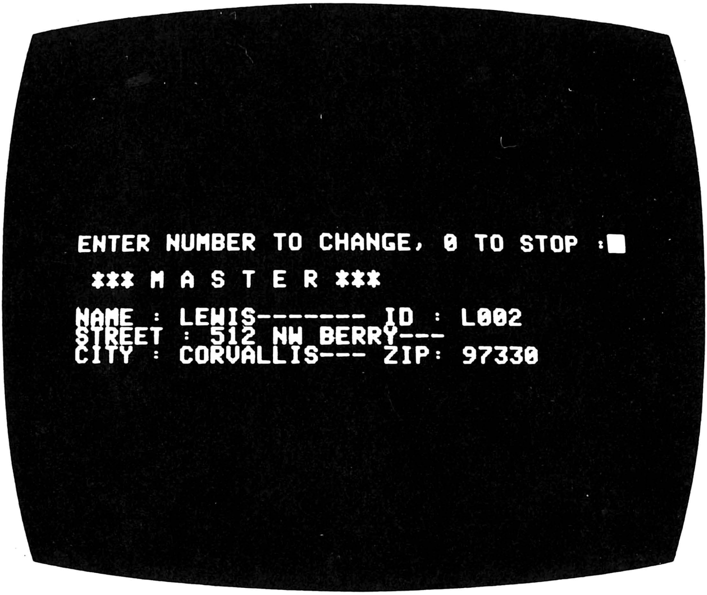
```
STRIKE <RTN> TO ENTER PAGE = 0█

     1 CHAPTERS IN ACCOUNTS
     CHAPTER [1] = MASTER
```

Figure 7.4 Starting at page #0



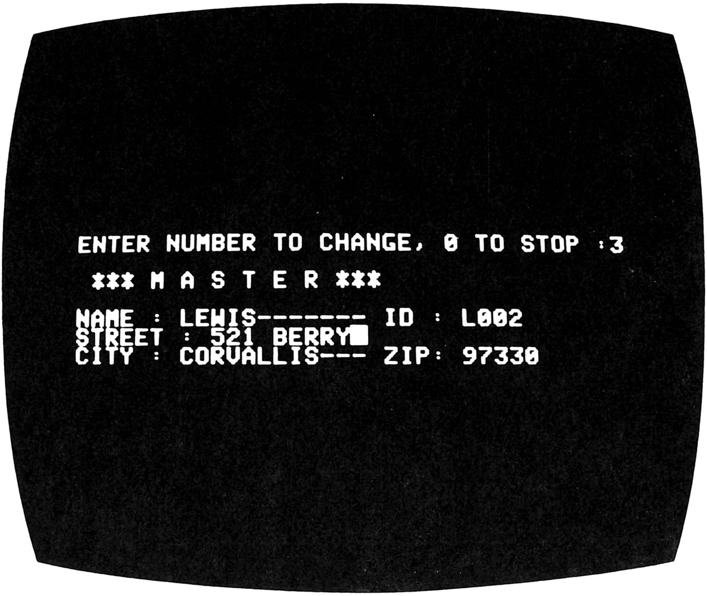
```
 *** M A S T E R ***

NAME : LEWIS-------- ID : L002
STREET : 512 NW BERRY---
CITY : CORVALLI█--- ZIP: -----
```

Figure 7.5 Enter data values in pre-defined form



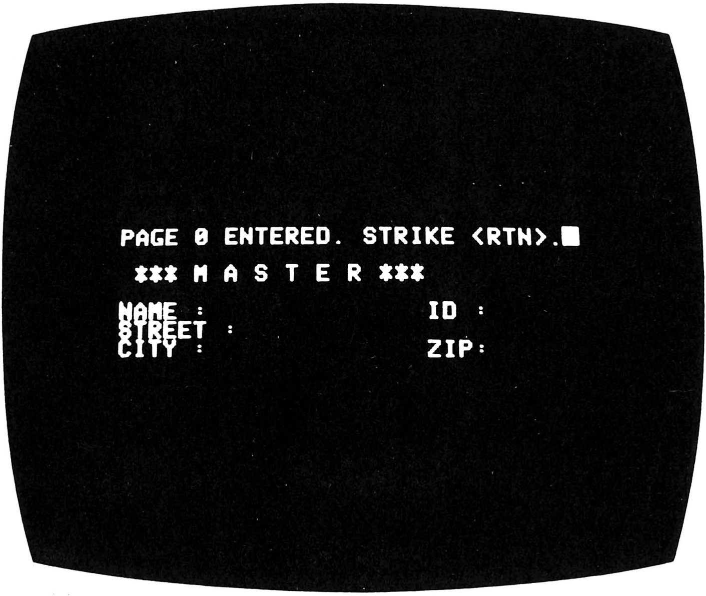
```
ENTER NUMBER TO CHANGE, 0 TO STOP :■
 *** M A S T E R ***
NAME : LEWIS-------- ID : L002
STREET : 512 NW BERRY---
CITY : CORVALLIS--- ZIP: 97330
```

Figure 7.6 Corrections?



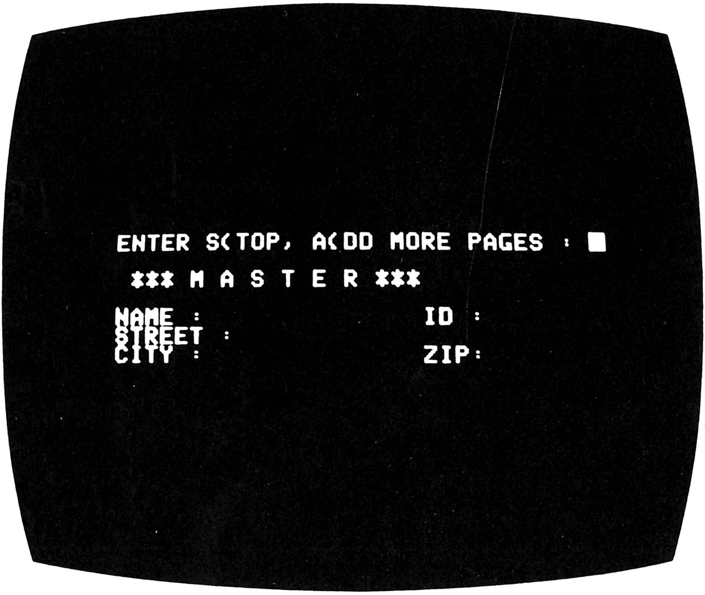
```
ENTER NUMBER TO CHANGE, 0 TO STOP :3
 *** M A S T E R ***
NAME : LEWIS-------- ID : L002
STREET : 521 BERRY■
CITY : CORVALLIS--- ZIP: 97330
```

Figure 7.7 Change data word #3



```
PAGE 0 ENTERED. STRIKE <RTN>.█
 *** M A S T E R ***
NAME :                  ID :
STREET :
CITY :                  ZIP:
```

Figure 7.8 Page is entered in chapter



```
ENTER S(TOP, A(DD MORE PAGES : █
 *** M A S T E R ***
NAME :                  ID :
STREET :
CITY :                  ZIP:
```

Figure 7.9 STOP or ADD more?

8

B(ROWSEing the Library

8.1 THE PURPOSE OF B(ROWSE

The B(ROWSE function accesses any MICROBOOK database, retrieves a set of data values, and displays them on the screen. There are two ways to access MICROBOOK files. The first one is through the data dictionary, and the second one is through the library. You must know and enter correct book titles, book passwords, chapter headings, and chapter passwords in order to browse a database in either case above. Figure 8.1 shows the tree structure of the BROWSE function system of prompts.

8.2 THE BROWSE FUNCTION COMMANDS

Refer to Figure 8.1:

(1) B(ooks: This command accesses MICROBOOK files by book title and chapter heading.
D(ata: This command accesses MICROBOOK files through the words in the data dictionary.
S(top: This command exits the current level and resumes at the level above it.

(2) O(pen: This command opens a book and a chapter. You must enter a book title, a book password, a chapter heading, and a chapter password.

(3) A(ny data word: This command retrieves the data words which are stored in the data dictionary.
N(ext data word: This command retrieves the next data word (in order) from the data dictionary.
F(orm: This command retrieves the form corresponding to the current data word (from Any data word or Next

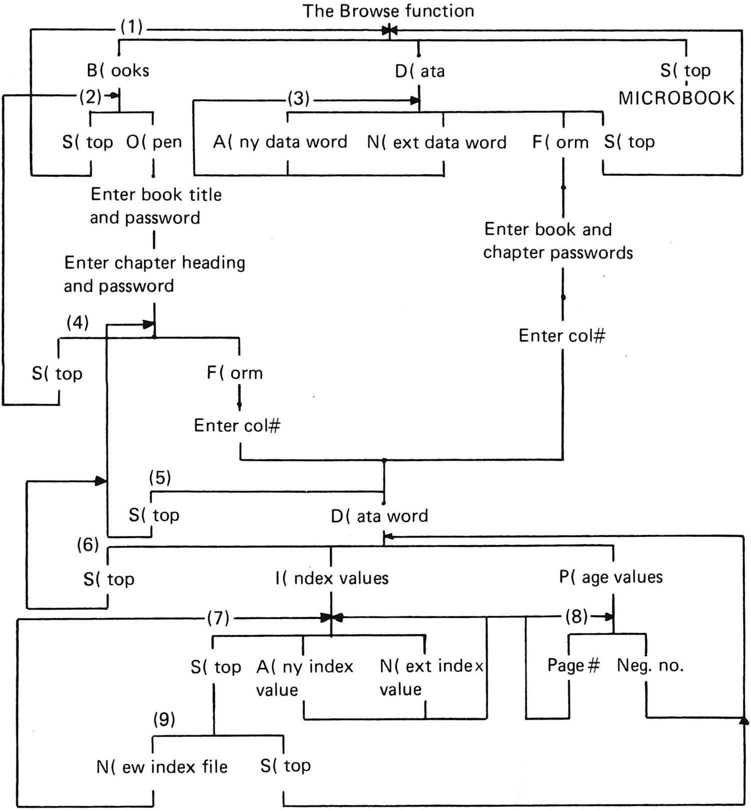

The Browse function
(1)
B(ooks
D(ata
S(top MICROBOOK
(2)
(3)
S(top
O(pen
A(ny data word
N(ext data word
F(orm
S(top
Enter book title and password
Enter chapter heading and password
Enter book and chapter passwords
Enter col#
(4)
S(top
F(orm
Enter col#
(5)
S(top
D(ata word
(6)
S(top
I(ndex values
P(age values
(7)
(8)
S(top
A(ny index value
N(ext index value
Page#
Neg. no.
(9)
N(ew index file
S(top

Figure 8.1 The browse function system of prompts tree structure

data word commands) and displays it on the screen. You must enter book and chapter passwords in which the data word is contained. If the chapter has more than one column, you must also enter the column number.

(4) F(orm: This command retrieves the form corresponding to a selected book and chapter (from Open command) and displays it on the screen.

(5) D(ata word: This command retrieves the data words which are in the form and displays them on the screen.

(6) I(ndex values: This command retrieves a page of values using an index value. You must enter the data word used to access the page.

P(age values: This command retrieves a page of values using a page number. The first page is page zero.

(7) A(ny index value: This command displays a page containing the index value you entered.
N(ext index value: This command displays a page containing the next index value in order.
(8) Page#: You enter a page number to display a page on the screen,
Negative number: You enter a negative number to stop displaying page values. Execution resumes from (6).
(9) N(ext index file: This command is used to change the name of the access word used to retrieve pages.

The B(ooks function is hierarchical because execution must pass through the root prompt (the B function) and succeeding levels of pre-defined prompts. A set of conditions is tested and satisfied for each prompt before execution moves to the next level. All tests of each prompt are inherited by subordinate ones.

8.3 EXCEPTION CONDITIONS

The B function cannot proceed if a test fails. For example, B(ROWSE will not continue if any of the following occur:

(1) No book in the library
(2) No data word in the data dictionary
(3) No chapter in the book
(4) Wrong book title
(5) Wrong book password
(6) Wrong chapter heading
(7) Wrong chapter password
(8) End of file

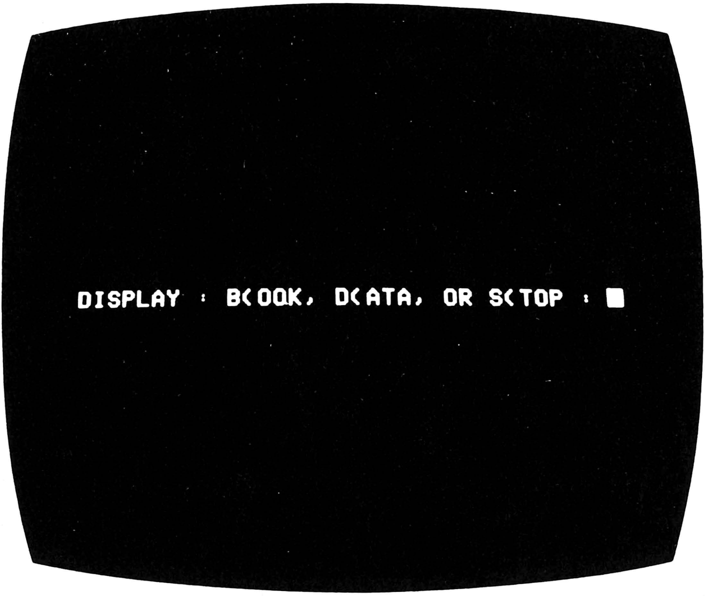
If one of the errors occurs, an error message is displayed on the screen and execution resumes from the previous step.

8.4 AN EXAMPLE

A sequence of figures illustrates the steps taken to browse through book ACCOUNTS. The figures do not show all steps, e.g., the password checks.

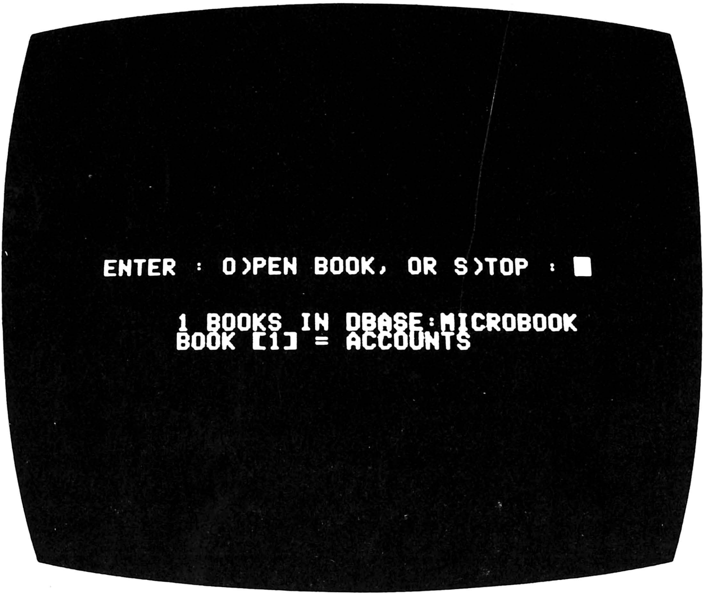
Figure 8.2 shows how to select access to the library through a book. Figure 8.3 shows how to open the book and provide a password. Figures 8.4 and 8.5 show how to enter the title of the book (ACCOUNTS) and the chapter name (MASTER).

The sequence of Figures 8.6, 8.7, and 8.8 shows how to retrieve and display the form for chapter MASTER along with its data words. Figures 8.9 and 8.10 show how to retrieve page #2 from the chapter. Figures 8.11 and 8.12 show how to delete page #2.



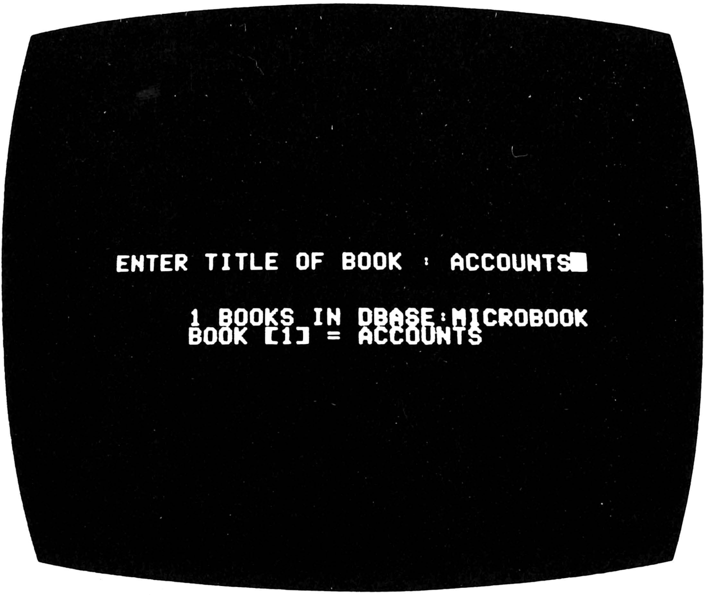
```
DISPLAY : B(OOK, D(ATA, OR S(TOP : ■
```

Figure 8.2 Select B(OOK



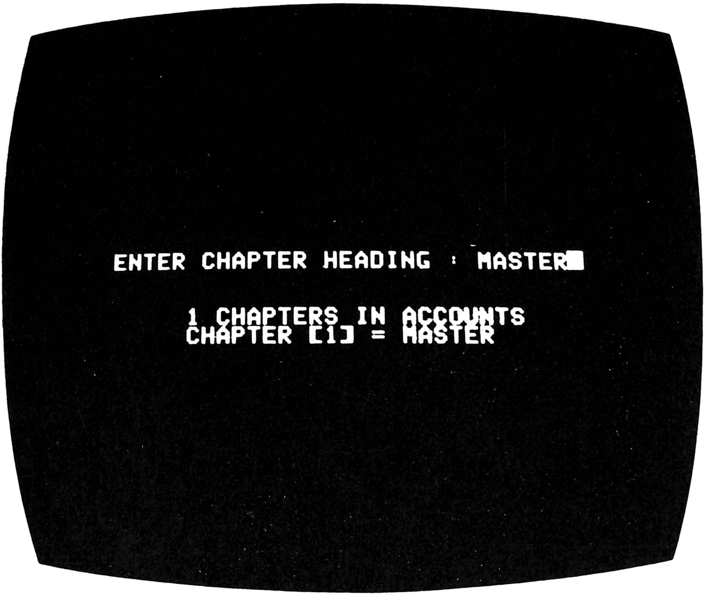
```
ENTER : O)PEN BOOK, OR S)TOP : ■

        1 BOOKS IN DBASE:MICROBOOK
        BOOK [1] = ACCOUNTS
```

Figure 8.3 Select O(PEN



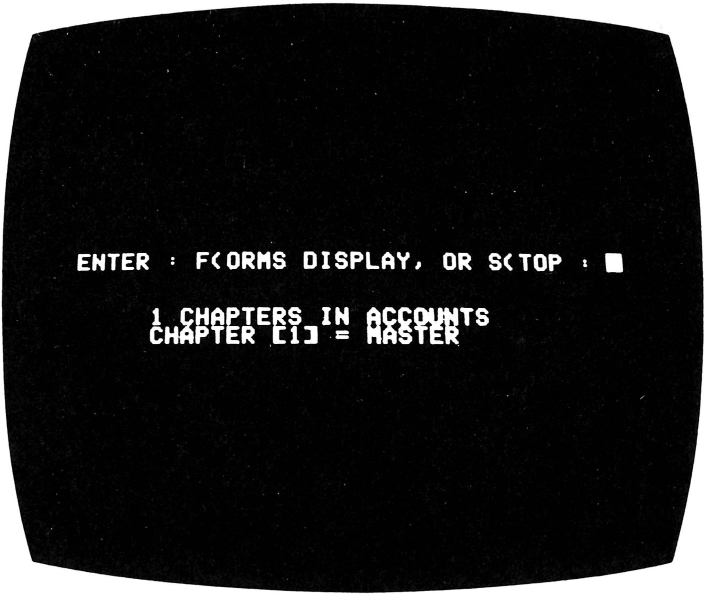
```
ENTER TITLE OF BOOK : ACCOUNTS█

          1 BOOKS IN DBASE:MICROBOOK
          BOOK [1] = ACCOUNTS
```

Figure 8.4 Enter ACCOUNTS



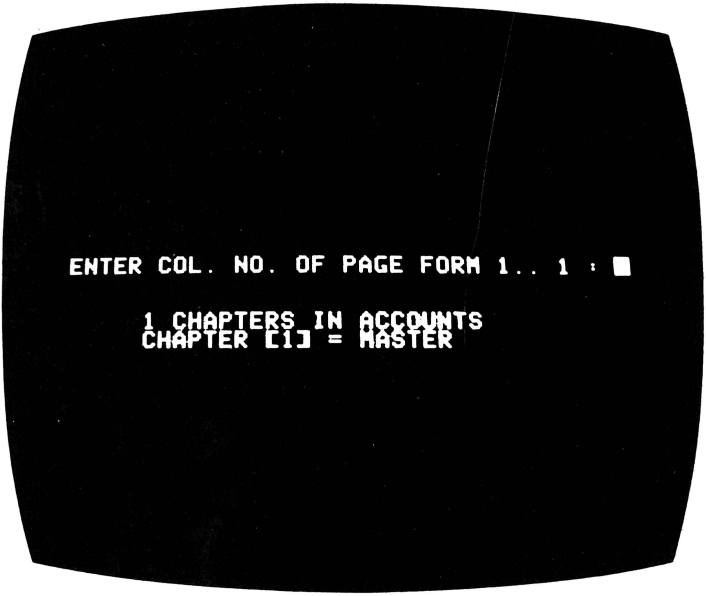
```
ENTER CHAPTER HEADING : MASTER█

          1 CHAPTERS IN ACCOUNTS
          CHAPTER [1] = MASTER
```

Figure 8.5 Enter chapter name



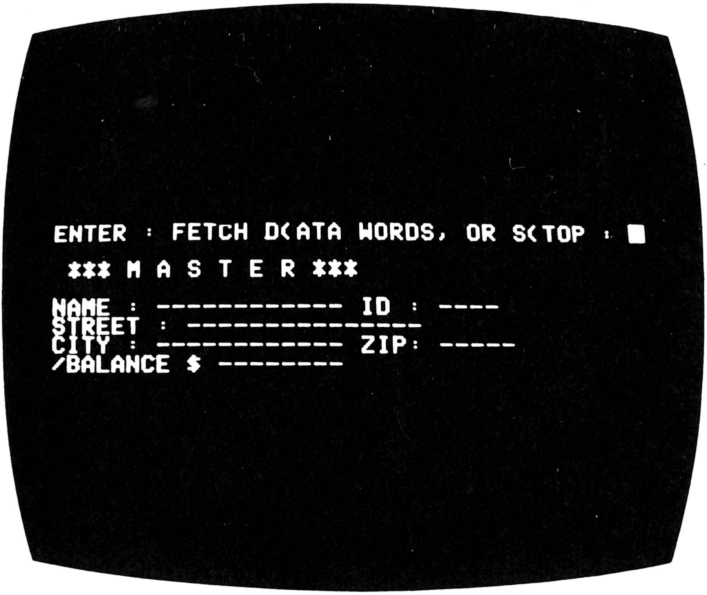
```
ENTER : F(ORMS DISPLAY, OR S(TOP : ■

         1 CHAPTERS IN ACCOUNTS
         CHAPTER [1] = MASTER
```

Figure 8.6 Select F(ORMS



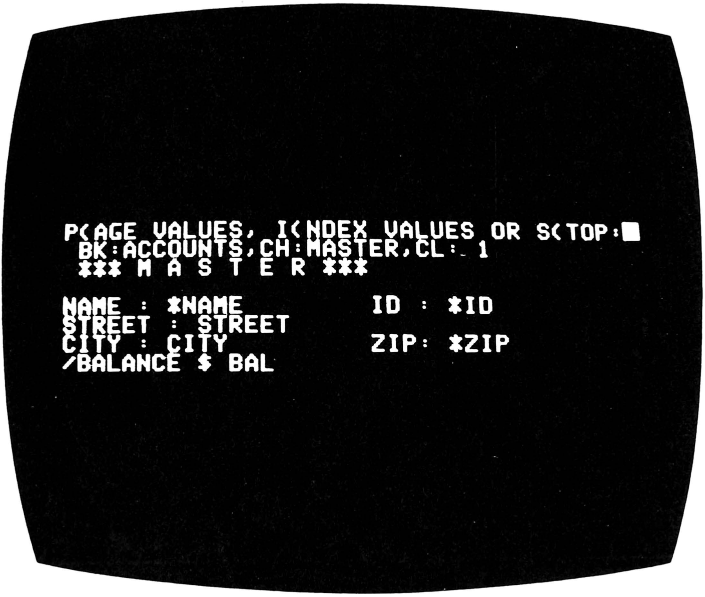
```
ENTER COL. NO. OF PAGE FORM 1.. 1 : ■

         1 CHAPTERS IN ACCOUNTS
         CHAPTER [1] = MASTER
```

Figure 8.7 Enter 1 column



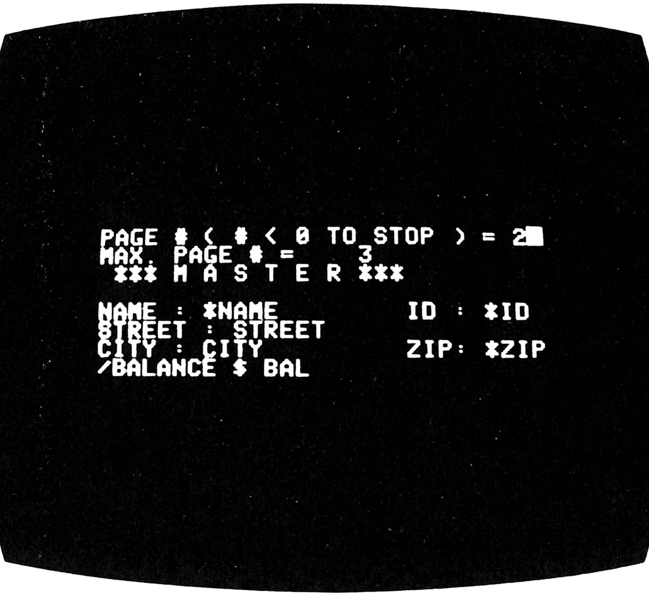
```
ENTER : FETCH D(ATA WORDS, OR S(TOP : █
 *** M A S T E R ***
NAME : -------------- ID : ----
STREET : -----------------
CITY : -------------- ZIP: -----
/BALANCE $ ---------
```

Figure 8.8 Select D(ATA



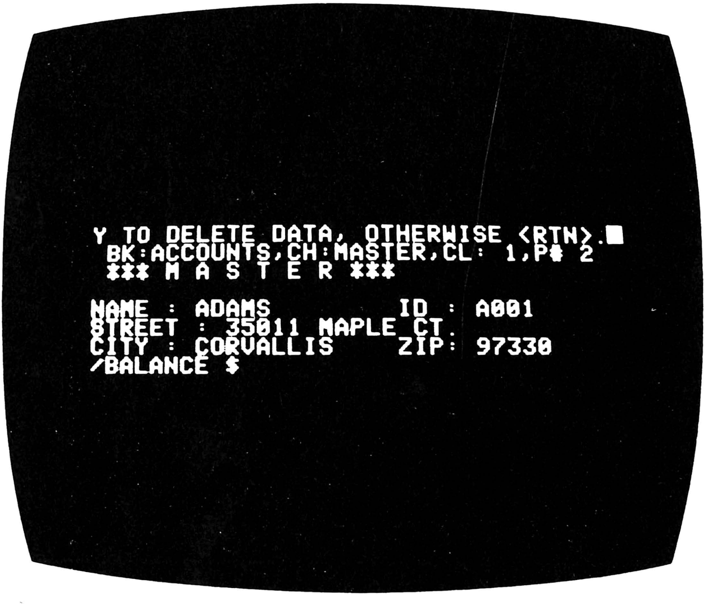
```
P(AGE VALUES, I(NDEX VALUES OR S(TOP:█
 BK:ACCOUNTS,CH:MASTER,CL:_1
 *** M A S T E R ***
NAME : *NAME           ID : *ID
STREET : STREET
CITY : CITY            ZIP: *ZIP
/BALANCE $ BAL
```

Figure 8.9 Select P(AGE



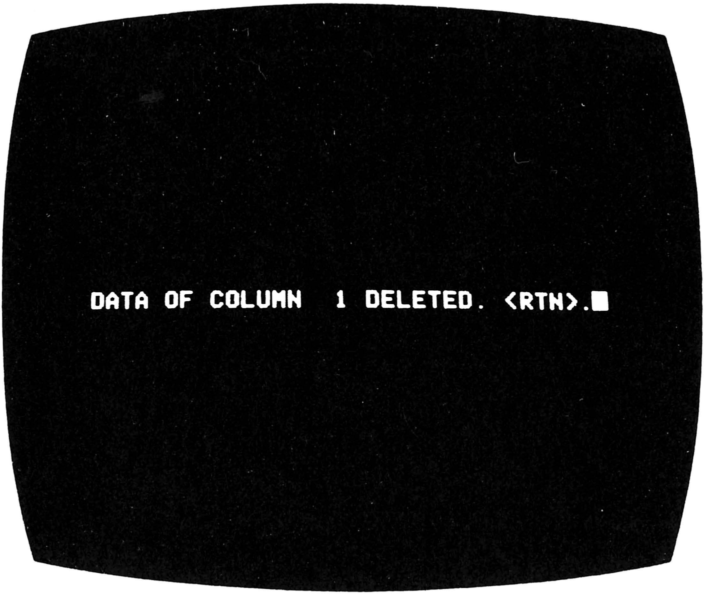
```
PAGE # ( # < 0 TO STOP ) = 2█
MAX. PAGE # =      3
 *** M A S T E R ***

NAME : *NAME        ID : *ID
STREET : STREET
CITY : CITY         ZIP: *ZIP
/BALANCE $ BAL
```

Figure 8.10 Retrieve page #2



```
Y TO DELETE DATA, OTHERWISE <RTN>.█
 BK:ACCOUNTS,CH:MASTER,CL: 1,P# 2
 *** M A S T E R ***

NAME : ADAMS        ID : A001
STREET : 35011 MAPLE CT.
CITY : CORVALLIS    ZIP: 97330
/BALANCE $
```

Figure 8.11 Delete page #2



```
DATA OF COLUMN  1 DELETED. <RTN>.█
```

Figure 8.12 Deletion done

The second line of prompt information in Figure 8.11 tells you the book (BK:) chapter (CH:), column (CL:), and page (P:) number of the displayed page. In the example

BK: ACCOUNTS CH: MASTER CL: 1 P#2

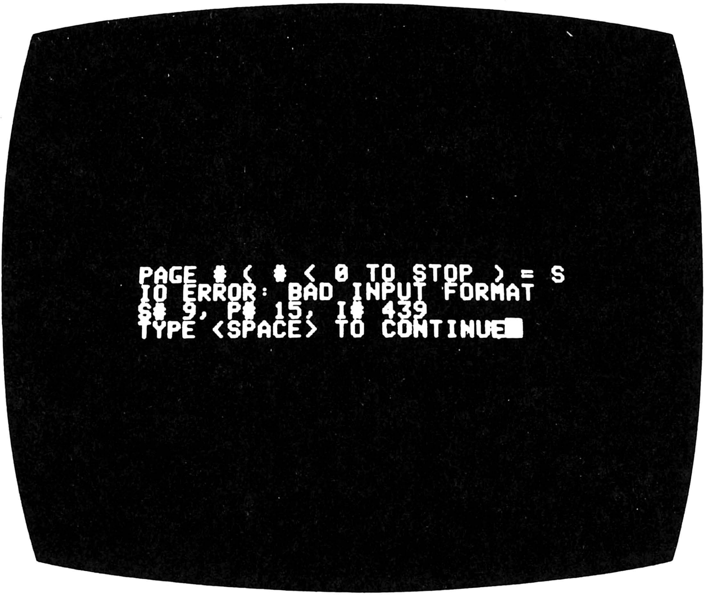
refers to book ACCOUNTS, chapter MASTER, column 1 of page #2.

The B(ROWSE function will not tolerate incorrect inputs as shown in Figure 8.13. When this happens you must press the space bar to continue. Then MICROBOOK will restart at the Command level. To get back to the B(ROWSE function, reenter B and make the selections shown in the figures again.

Figures 8.14 and 8.15 show how to alternately access a page via a data word. When the prompt

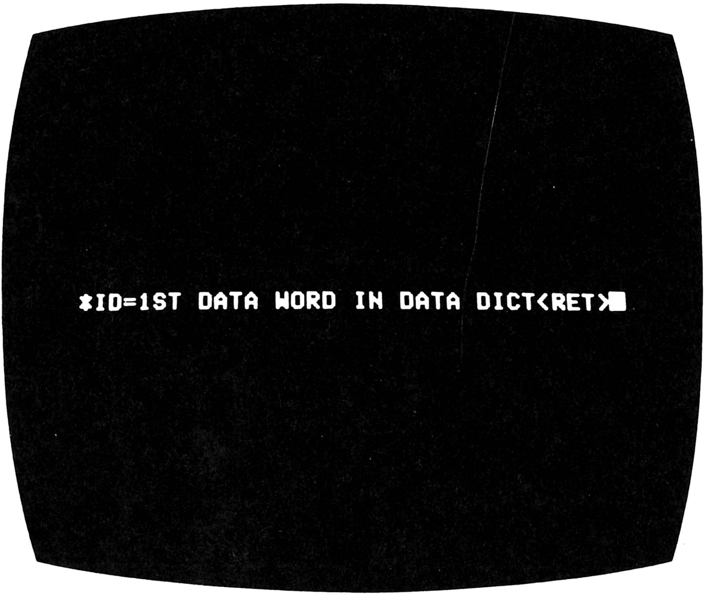
DISPLAY : B(OOK, D(ATA, or S(TOP :

is given as shown in Figure 8.2, and you enter D instead of B, the first data word in the data dictionary is displayed. Subsequently, entry of book and chapter passwords will allow access to a page containing this data value.



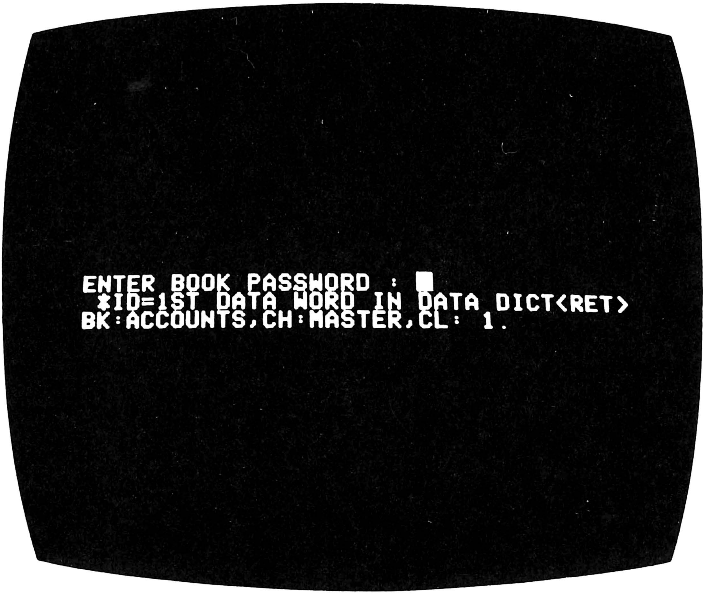
```
PAGE # ( # < 0 TO STOP ) = S
IO ERROR: BAD INPUT FORMAT
S# 9, P# 15, I# 439
TYPE <SPACE> TO CONTINUE█
```

Figure 8.13 Error recovery



```
*ID=1ST DATA WORD IN DATA DICT<RET>█
```

Figure 8.14 Selection of D(ATA instead of B(OOK



```
ENTER BOOK PASSWORD : █
 *ID=1ST DATA WORD IN DATA DICT<RET>
BK:ACCOUNTS,CH:MASTER,CL: 1.
```

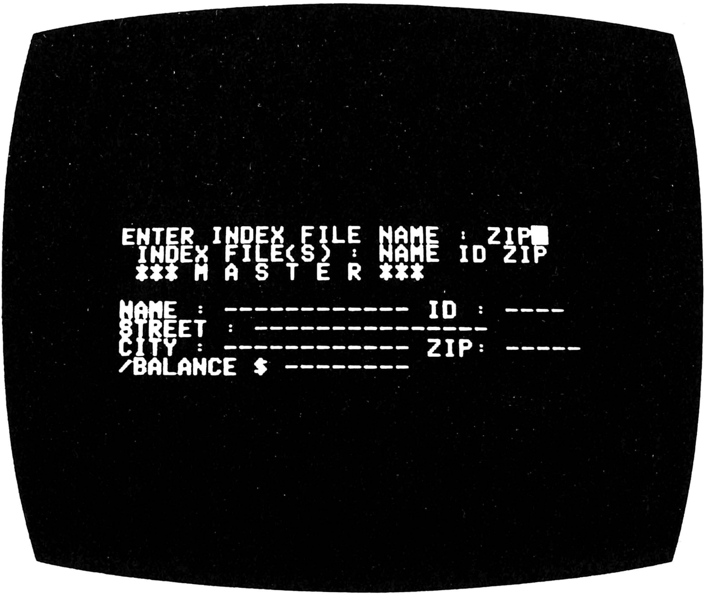
Figure 8.15 Enter passwords to display a page

Figure 8.16 shows how to make access via key word values instead of by page number. This mode of access is activated by selecting I when the following prompt is given:

P(AGE VALUES, I(NDEX VALUES OR S(TOP

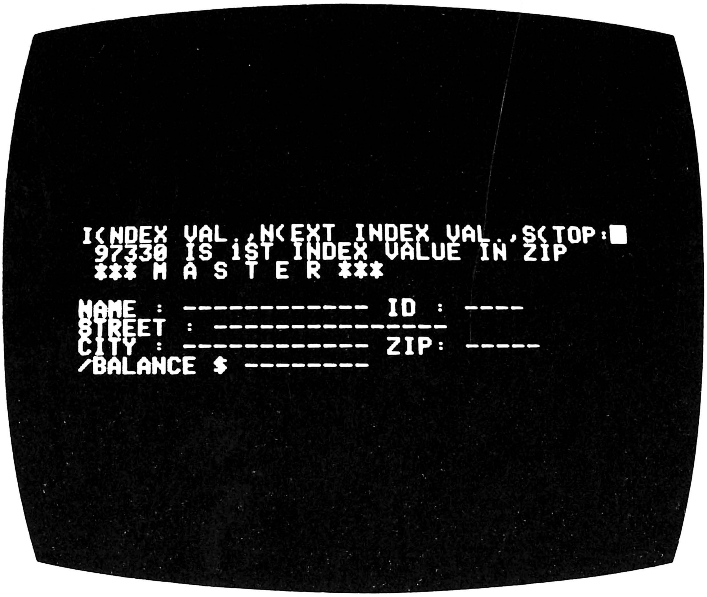
Figure 8.17 shows how to specify a data word value. The page containing ZIP=97330 will be retrieved and displayed on the screen. If more than one page contains 97330, then each page is selected, one at a time, by selecting N(EXT INDEX VAL. When all values have been displayed the message shown in Figure 8.19 will be displayed.

Figure 8.20 shows what happens when a retrieval fails due to a deletion. Figure 8.21 shows the response from MICROBOOK when no match occurs.



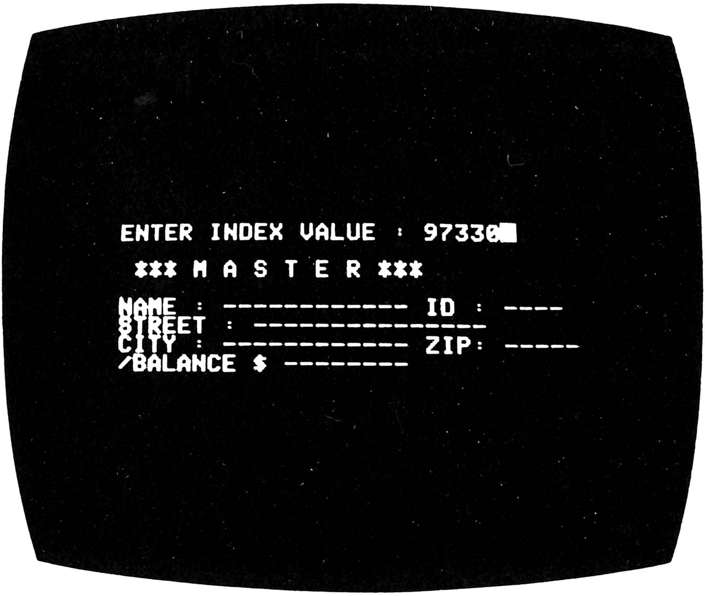
```
ENTER INDEX FILE NAME : ZIP█
 INDEX FILE(S) : NAME ID ZIP
 *** M A S T E R ***

NAME : ------------ ID : ----
STREET : ---------------
CITY : ------------- ZIP: -----
/BALANCE $ --------
```

Figure 8.16 Access via key word



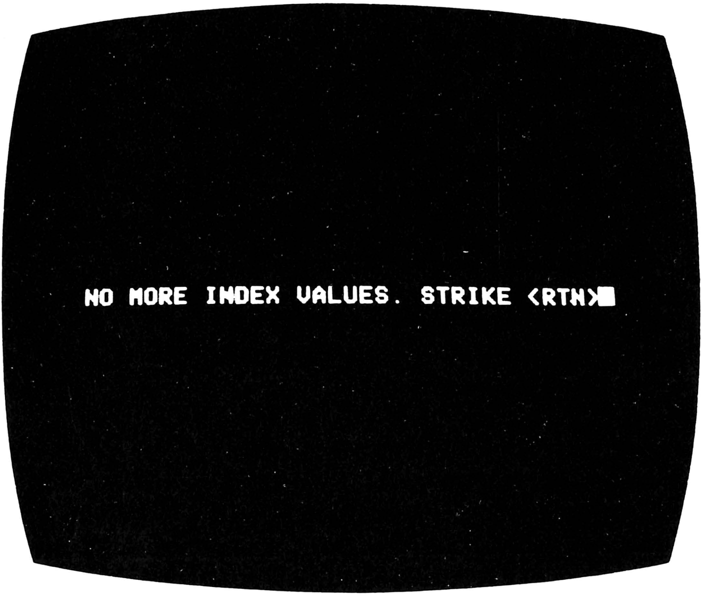
```
I(NDEX VAL.,N(EXT INDEX VAL.,S(TOP:█
 97330 IS 1ST INDEX VALUE IN ZIP
 *** M A S T E R ***

NAME : ------------ ID : ----
STREET : ---------------
CITY : ------------- ZIP: -----
/BALANCE $ --------
```

Figure 8.17 Select I(NDEX



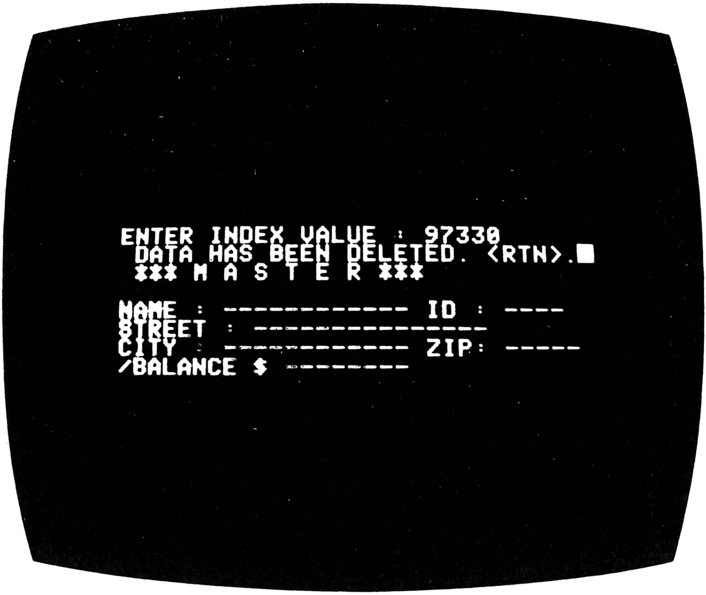
```
ENTER INDEX VALUE : 97330█
 *** M A S T E R ***
NAME : ------------ ID : ----
STREET : ---------------
CITY : ------------ ZIP: -----
/BALANCE $ --------
```

Figure 8.18 Access all pages with ZIP=97330



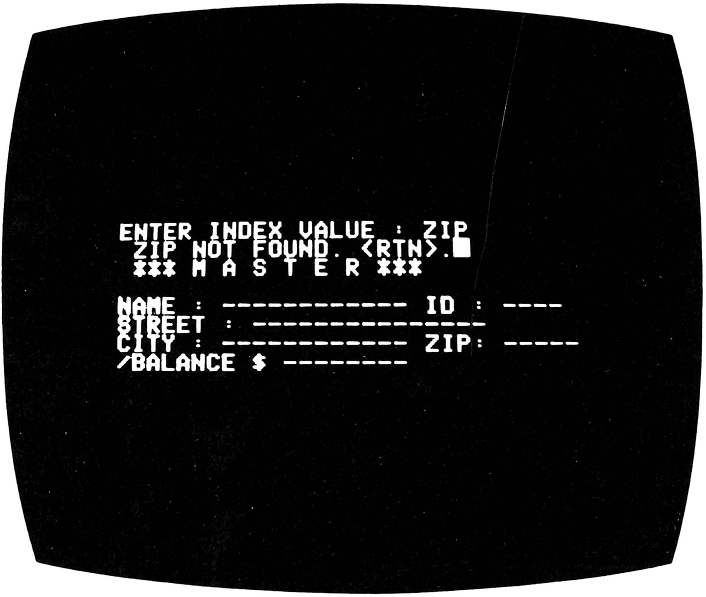
```
NO MORE INDEX VALUES. STRIKE <RTN>█
```

Figure 8.19 End of chapter reached



```
ENTER INDEX VALUE : 97330
 DATA HAS BEEN DELETED. <RTN>.█
 *** M A S T E R ***

NAME : ------------ ID : ----
STREET : --------------
CITY : ------------ ZIP: -----
/BALANCE $ --------
```

Figure 8.20 Retrieved deleted page



```
ENTER INDEX VALUE : ZIP
 ZIP NOT FOUND. <RTN>.█
 *** M A S T E R ***

NAME : ------------ ID : ----
STREET : --------------
CITY : ------------ ZIP: -----
/BALANCE $ --------
```

Figure 8.21 No match with value

9

EX(ECUTEing the QUERY Processor

9.1 The Purpose of QUERY

The purpose of QUERY is to direct the computer to process information stored in a MICROBOOK library. That is, QUERY can be instructed to copy information from one page to another, calculate numerical results, and display a page at a time on the screen.

QUERY is a high level, interpretive language processor that lets you develop application *procedures* without an in-depth knowledge of computer programming.

9.2 How to Use QUERY

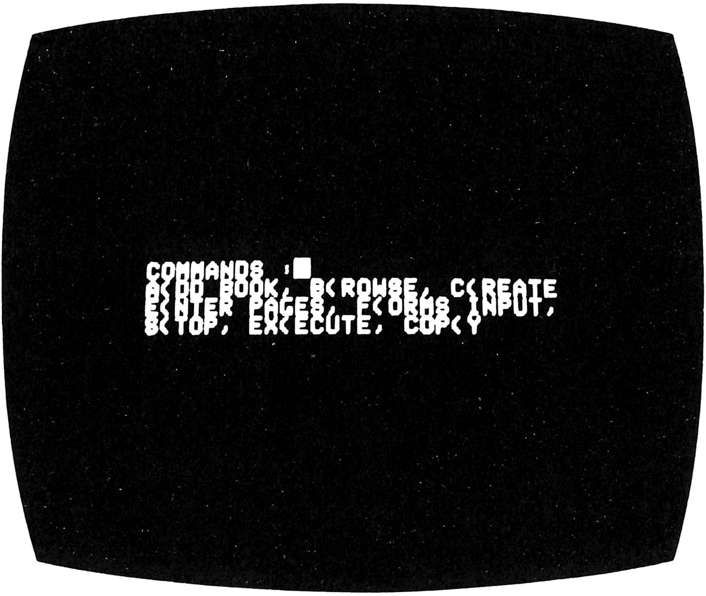
The QUERY processing program is accessed from MICROBOOK's Command level by selecting the EX(ECUTE function followed by entering "QUERY" as shown in Figures 9.1 and 9.2. The system will take 15-20 seconds to prepare for a query command.

Figure 9.3 shows the screen after QUERY begins executing. Entering G, S, N, R, or Q causes the following to occur:

G(ET : If a previously entered sequence of QUERY statements were stored in a file, then this function retrieves the sequence and displays it on the screen. This sequence becomes the current sequence of statements.

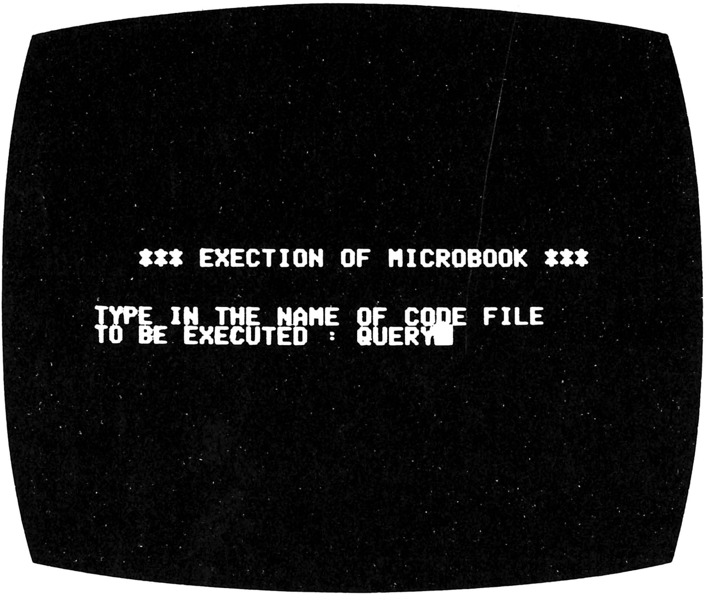
S(AVE : The *current* sequence of QUERY statements is saved in a file under a name you provide.

N(EW : A new sequence of QUERY statements is created. The previous sequence will be erased, so be sure to S(AVE it first if it is needed later.



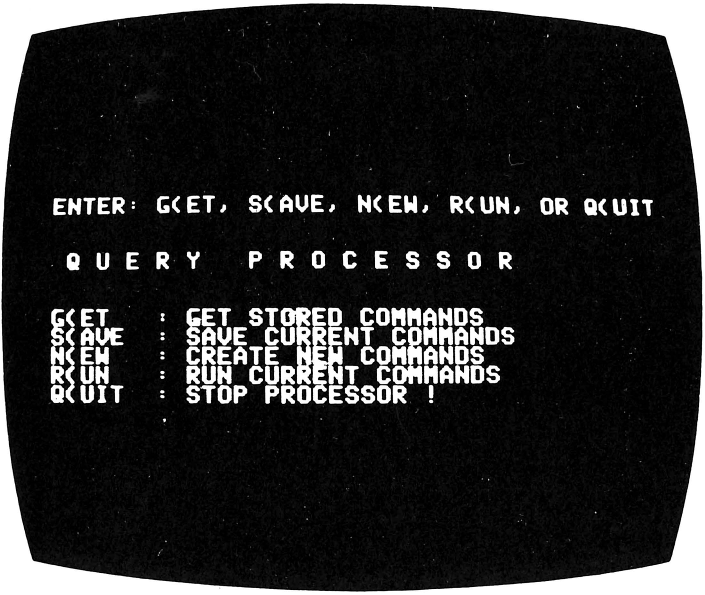
```
COMMANDS :
A(DD BOOK, B(ROWSE, C(REATE
E(NTER PAGES, F(ORMS INPUT,
S(TOP, EX(ECUTE, COP(Y
```

Figure 9.1 Select X command



```
*** EXECTION OF MICROBOOK ***

TYPE IN THE NAME OF CODE FILE
TO BE EXECUTED : QUERY
```

Figure 9.2 Enter "QUERY" to get the QUERY processor



```
ENTER: G(ET, S(AVE, N(EW, R(UN, OR Q(UIT

 Q U E R Y   P R O C E S S O R

G(ET    : GET STORED COMMANDS
S(AVE   : SAVE CURRENT COMMANDS
N(EW    : CREATE NEW COMMANDS
R(UN    : RUN CURRENT COMMANDS
Q(UIT   : STOP PROCESSOR !
```

Figure 9.3 QUERY functions

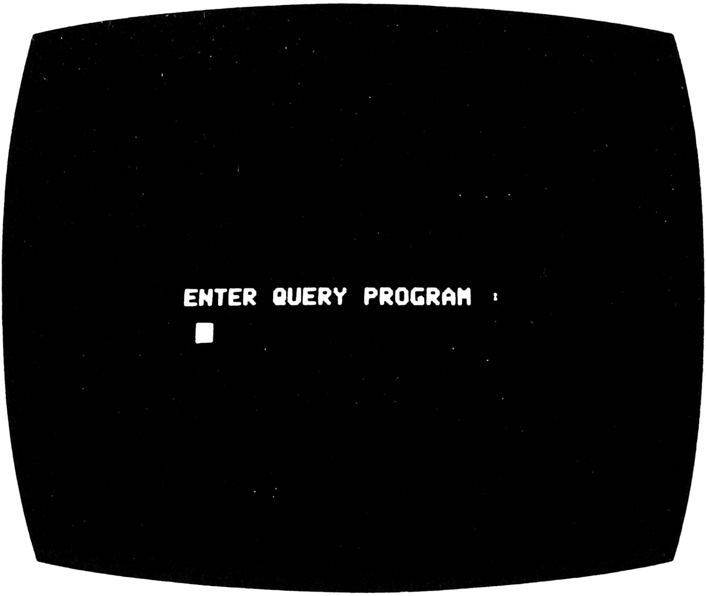
R(UN : The current sequence of QUERY statements is executed. Furthermore, passwords to selected books and chapters are requested whenever needed.

Q(UIT: Halt the QUERY processor and return control to the Command level of MICROBOOK.

Figure 9.4 shows how to enter a sequence of QUERY statements. Selection of the N(EW function, above, causes the previous sequence of statements to be erased. A new sequence of statements can then be entered.

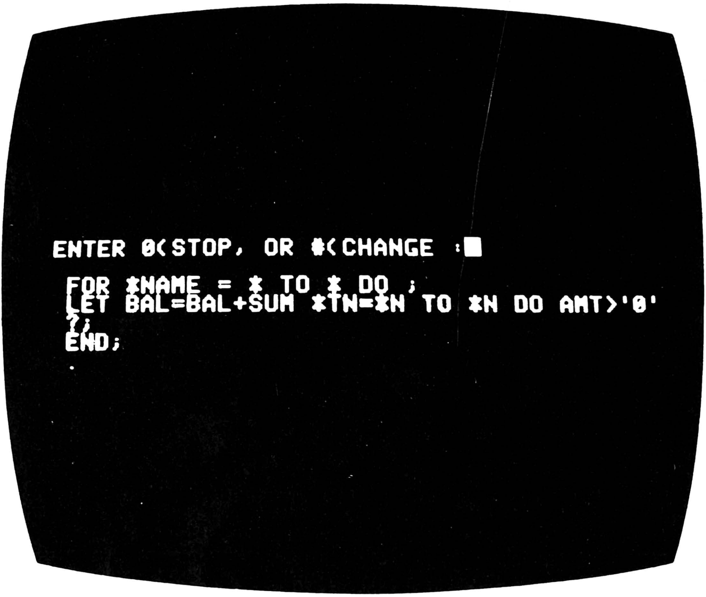
Figure 9.5 illustrates a newly entered sequence of QUERY statements. You can edit this sequence. For example, if you enter zero, the sequence becomes the *current* sequence. If you enter a line number, then the corresponding statement or line can be re-entered.

When a new sequence of statements is entered and no more changes made, the QUERY processor displays the menu shown in Figure 9.3. Suppose the current sequence is saved in a file named QSAVE. Figure 9.6 shows how to use the S(AVE function to save the current sequence. To retrieve QSAVE, use the G(ET function.



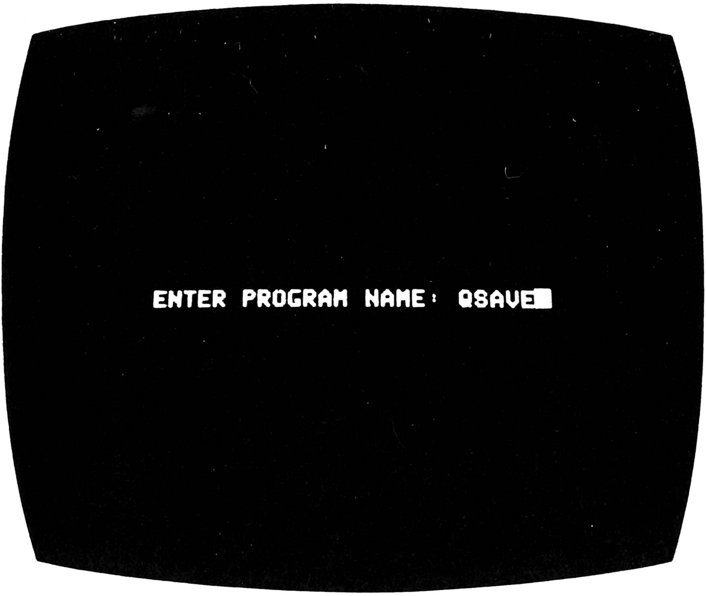
```
ENTER QUERY PROGRAM :
 ■
```

Figure 9.4 N(EW sequence



```
ENTER 0<STOP, OR #<CHANGE :■

 FOR *NAME = * TO * DO ;
 LET BAL=BAL+SUM *TN=*N TO *N DO AMT>'0'
 ?;
 END;
 .
```

Figure 9.5 Sample sequence of QUERY commands



```
ENTER PROGRAM NAME: QSAVE
```

Figure 9.6 S(AVE function

9.3 THE QUERY LANGUAGE

The QUERY language processor expects input as follows: see Figure 9.5 for example:

(1) The last line must be a period (in the first column).
(2) Every statement in the sequence must end with a semicolon.
(3) Statements must be less than 160 characters long. They can be broken up into several lines on the screen.

The last rule for forming QUERY statements means that a very long statement can be entered as more than one line. The right-hand margin of the screen is marked with an "!" character to delimit the width of the screen. For example, the sequence of statements in Figure 9.5 can be entered as shown below:

```
FOR *NAME=* TO *
    DO ;
    LET BAL=BAL+SUM
            *TN=*N TO *N DO
            AMT>'0' ;
    ? ;
END ;
.
```

The QUERY language processor ignores blanks; it accepts indentation and imbedded blanks. This helps to make the sequence more readable.

The example sequence above can be expressed in English, as follows:

"For every page containing a name in data word *NAME, do the following operations:

(1) SUM the values of AMT, which are greater than zero, in the pages that have data words *TN equal to *N.
(2) Add the value of the SUM from (1) above to the value of BAL.
(3) Store the sum from (2) above in the data value corresponding to BAL.
(4) Display (?;) the page containing the updated BAL and value of *NAME.

The processing halts when all values of *NAME have been processed."

This example can be modified to compute a new balance, BAL, from the sum of all outstanding transactions, *TN, for each *NAME, as follows:

```
FOR *NAME=* TO *
     DO ;
     LET BAL=BAL+SUM *TN=*N TO *N
                DO AMT=AMT;
     ? ;
END ;
•
```

Hence, the illustration above performs the posting phase of an accounts receivable system in four QUERY statements! The QUERY language is a very high level language, indeed.

9.4 STRUCTURE OF THE QUERY LANGUAGE

There are five statement types in QUERY:

(1) FOR. This statement is used to retrieve pages via their key words.
(2) LET. This statement is used to perform simple copy and arithmetic operations.
(3) ?. This statement is used to display an entire page (one column) at a time.
(4) END. This statement terminates a FOR statement.

(5) • This is not really a statement, but is used to specify the end of a sequence of QUERY statements.

FOR-Statements

The forms of a FOR-statement are as follows:

FOR $\alpha=\beta$ TO δ DO ;
FOR $\alpha=\beta$ TO δ DO ρ <op> δ

where α is any key word taken from the data dictionary
β is any data value stored in a page indexed by α.
δ is any data word stored in a page indexed by α and is "after" β in alphabetical or numerical order.
ρ is any data word stored in a page indexed by α and is used to restrict access by <op>.
δ is a data word or literal constant.
<op> is one of the relational operators: = equals; < > not equal to; > greater than; > = greater than or equal to; < less than; < = less than or equal to.

In addition, α must be prefixed by * to indicate that it is a key word. Furthermore, when * is used in place of β it means to use the first data value of α. When * is used in place of δ it means to use the last data value of α.

All *constant* data values must be enclosed in single quotes to distinguish them from data words. For example:

FOR *NAME='LEWIS' to 'SMITH' DO ;

means to retrieve all pages indexed by *NAME beginning with LEWIS and ending with SMITH. However,

FOR *NAME='LEWIS' to 'ADAMS' DO ;

would fail because ADAMS comes before LEWIS in alphabetical order. Another example:

FOR *NAME='LEWIS' TO * DO BAL > '0' ;

means to retrieve all pages with *NAME equal to LEWIS, then SMITH, etc. until the last page has been retrieved, but to skip all pages containing BAL not greater than zero.

END-Statement

The END-statement delimits or terminates the FOR-statement. FOR-statements *cannot* be nested. Therefore, if two FOR-statements are in a sequence, the END-statement that terminates the first FOR-statement appearing in the sequence must occur before the second FOR-statement.

This is legal:

```
FOR *NAME=* TO * DO ;
? ;
END ;
FOR *TN=* TO * DO ZIP <> '97330' ;
? ;
END ;
```

●

But this is *not* legal:

```
FOR *NAME=* TO * DO ;
  FOR *TN=* TO * DO ZIP <> '97330' ;
  ? ;
  END ;
END ;
```

●

LET-Statements

Every LET-statement must appear within a FOR-END pair of statements. The forms of a LET-statement are

```
LET α=β ;
LET α=β+δ ;
LET α=β−δ ;
LET α=β * δ ;
LET α=β / δ ;
```

The first form moves the contents of β to the contents of α. The next four forms perform the usual arithmetic operations on β and δ data values. These data values must be numeric.

The data words α, β, and δ must be valid data words within a page retrieved by an enclosing FOR-END pair unless they are prefixed by #. The values of β and δ may be constants if enclosed by single quotes. Some examples are

```
FOR *NAME=* TO * DO ;
  LET BAL='0' ;
  LET BAL='2' * BAL ;
  LET BAL='3'+'5' ;
  LET BAL=ZIP+BAL ;
  LET BAL=BAL / '2' ;
END ;
```

●

The # prefix can be used to output values to another book/chapter which is *not* indexed by an enclosing FOR-END pair. For example, suppose AMT is a data word defined in another chapter. Then,

```
FOR *NAME=  * TO * DO ;
  LET #AMT=BAL ;
END ;
●
```

will move the value stored in BAL to the field defined by AMT starting at page zero. Each time the FOR-END pair is executed, the page number for AMT is incremented. Thus, if FOR-END is executed 10 times, pages 0 through nine are filled with values from BAL. Details of this feature are shown in action in the Accounts Receivable example.

Aggregates

In addition, δ (and β if using the first form) may be an *aggregate expression.* The built-in aggregate expressions of QUERY are

SUM : sum to a total
AVG : average value
COUNT: enumerate (how many)
MIN : find the smallest value
MAX : find the largest value

The form of an aggregate expression is as follows:

<AGG> $\alpha=\beta$ TO δ DO ρ <op> σ ;

where AGG is SUM, AVG, COUNT, MIN or MAX

α is any key word prefixed with *
β is a literal or key word or data word, or *
δ is a literal or data word or *
ρ is a data word
σ is a data word or literal
<op> is a relational: = equals
<> not equal to
> greater than
>= greater or equal
< less than
<= less or equal

Some examples of aggregate expressions are:

```
SUM *N= * TO * DO AMT=AMT ;
```

This means to sum the AMT values of every page indexed by key word *N.

```
AVG *NAME='LEWIS' TO * DO BAL > '10.0' ;
```

This means to compute the average BAL value greater than 10.00 in the pages indexed by *NAME from LEWIS to the end of the chapter.

```
COUNT *NAME=* TO 'LEWIS' DO BAL > '0' ;
```

This means to count the number of BAL values greater than zero in the pages indexed by *NAME from beginning to (and including) the page indexed by LEWIS.

```
MAX *ZIP=* TO * DO BAL=BAL ;
```

This means to compute the largest BAL in all pages indexed by *ZIP.

The aggregate expression can be used as part of a LET-statement. For example, the SUM value can be stored in a page indexed via a FOR-END pair as shown below.

```
FOR *NAME='LEWIS' TO 'LEWIS' DO ;
  LET BAL=SUM *ZIP= * TO * DO BAL=BAL ;
END ;
•
```

The aggregate expression can also be used alone in the display statement.

?-Statement

The ?-statement is used to display the computed result of an aggregate expression or the page(s) retrieved for a FOR-END pair. The two forms are

```
? <AGGREGATE> ;
```

and

```
FOR <rest> ;
? ;
END ;
```

where,
<AGGREGATE> is an aggregate expression and <rest> is the remainder of a FOR-statement.

For example, to display the highest value of an outstanding balance, BAL, you can enter and R(UN the following *single* statement followed by a period.

```
? MAX *NAME=* TO * DO BAL=BAL ;
•
```

However, to display a page of data values

```
FOR *NAME='LEWIS' TO 'LEWIS' DO ;
? ;
END ;
•
```

is required because the second form must be enclosed within a FOR-END pair.

9.5 STATEMENT SUMMARY

The following statement summary lists all forms of QUERY statements. The notation is explained in the following list:

```
FOR α=β TO δ <op> σ ;
FOR α=β TO δ DO ;
END ;

? <AGGREGATE> ;

? ;

LET α=β ;
LET α=<AGGREGATE> ;
LET α=β <arith-op> δ ;
LET α=β <arith-op> <AGGREGATE> ;
```

or

```
LET #α=β ;
LET #α=<aggregate> ;
LET #α=β <arith-op> δ ;
LET #α=β <arith-op> <aggregate> ;
```

where,

<arith-op> is one of the following:
(1).+
(2).−
(3).*
(4)./

and AGGREGATE is of the following form:

<AGG> $\alpha=\beta$ TO δ TO ϱ <op> σ

where,

<AGG> is one of the following:

(1). SUM
(2). AVG
(3). MAX
(4). MIN
(5). COUNT

and <op> is one of the following:

(1).	=	equal
(2).	< >	not equal
(3).	>	greater than
(4).	<	less than
(5).	> =	greater or equal
(6).	< =	less or equal

The values of β, δ, and σ can be either constants (literals) enclosed in single quotes or data words. If <AGGREGATE> is enclosed within a FOR-END pair, then β and δ can also be identical to the name used in "FOR α". For example

```
FOR *NAME=* TO * DO ;
LET BAL=SUM *TN=*NAME DO AMT=AMT ;
END ;
•
```

Incorrect statements cause the QUERY processor to fail. When this occurs, MICROBOOK is reinitialized to the Command level. The error message

```
Error, expecting XXX, strike <rtn>
```

is usually given to indicate the problem. To continue, press the RETURN key.

9.6 ILLUSTRATIONS

Consider the following illustrative examples. In each case it is assumed that a book/chapter have been created corresponding to the key words, etc.

Example I

```
? COUNT *ZIP=* TO * DO ZIP > '97000' ;
FOR *ZIP=* TO * DO ;
? ;
END ;
•
```

This sequence causes two passes over all pages indexed by *ZIP. The first pass is carried out by the aggregate expression which COUNTs the number of pages with ZIP greater than 97000. The second pass displays each page in zipcode order.

The first access to the chapter containing these pages results in a request for you to enter the book and chapter passwords. Failure to provide a correct password will cause the QUERY procedure to fail.

Example II

```
FOR *NAME=* TO * DO ;
  LET BAL='0' ;
END ;
•
```

This sequence retrieves all pages accessed by *NAME (in alphabetical order), and sets BAL to zero. The first access to a page results in a password check on book and chapter.

Example III

```
FOR *ID=* TO * DO ;
  LET BAL=SUM *TD=*ID TO *ID DO AMT=AMT ;
  ? ;
END ;
•
```

Two chapters are scanned. The pages of the chapter containing key word *ID are retrieved, one at a time. Then, for each page, the data word BAL is updated with the SUM of the AMT values stored in a second chapter indexed by *TD. The SUM is carried out over all values of *TD which equal the value stored in *ID.

Finally, each updated page is displayed and all passwords are checked on the first access.

Example IV

```
FOR *NAME='LEWIS' TO 'LEWIS' DO ;
  LET AGE='39' ;
END ;
•
```

This sequence causes the AGE value of the page containing *NAME equal to LEWIS to be set to 39. All passwords are checked before access is allowed.

Example V

```
? COUNT *NAME=* TO *DO AGE > '39' ;
●
```

This causes the number of pages with AGE > '39' to be counted and then the count displayed. Passwords are checked.

10

COP(Ying the Library to Another Disk

10.1 THE PURPOSE OF COP(Y

The COP(Y function provides the following services for the user:

(1) Copies selected books, chapters, and pages from one disk volume onto a second disk volume.

(2) Removes deleted pages and compacts each chapter so that disk space is preserved.

(3) Allows you to extend the length of any chapter by entering a new upper limit to the maximum number of pages reserved.

These three goals are achieved in MICROBOOK by selectively copying from one disk volume to another. It is *not* possible to copy a library from one disk back onto the same disk.

10.2 HOW TO USE COP(Y

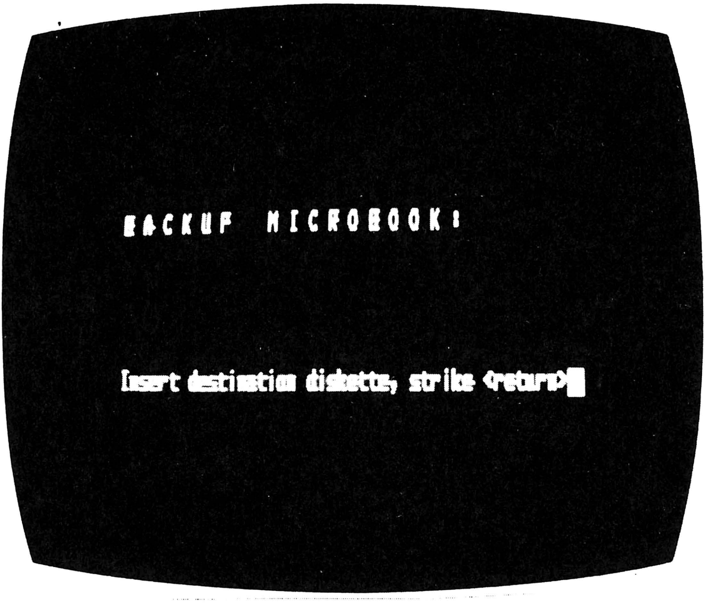
Type P at the Command level of MICROBOOK to enter the COP(Y function. After a pause, the message shown in Figure 10.1 will appear on the screen. Insert a destination disk in drive #1 (this requires removal of MICRO:, temporarily). The disk volumes of source (#2) and destination (#1) must have different names.

Type RETURN and the following prompt is displayed:

```
ENTER VOLUME NUMBER OR NAME:
```

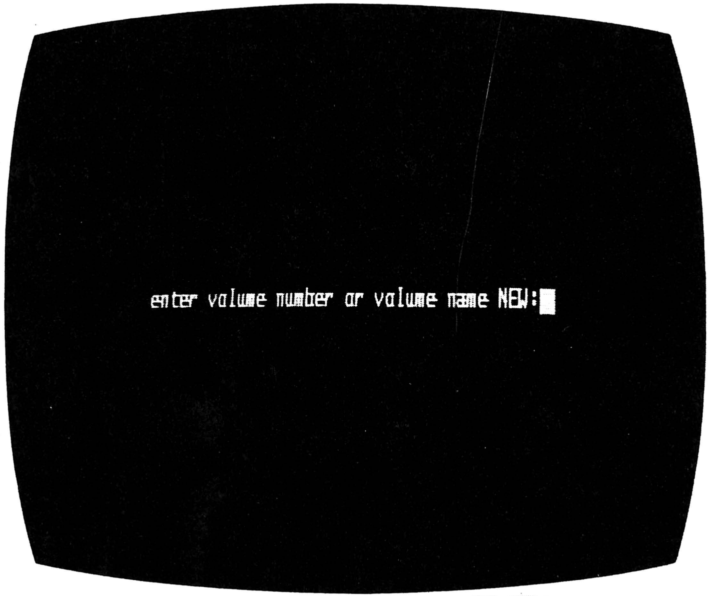
Type the volume name or number of the destination disk, remembering to terminate it with a colon, for example:

```
NEWDISK:
```



```
BACKUP MICROBOOK:

Insert destination diskette, strike <return>
```

Figure 10.1 Insert new volume into drive #1, and then strike RETURN



```
enter volume number or volume name NEW:
```

Figure 10.2 Enter name of new volume

Then press RETURN and when you are sure, enter 'Y' to the prompt: ARE YOU SURE?.

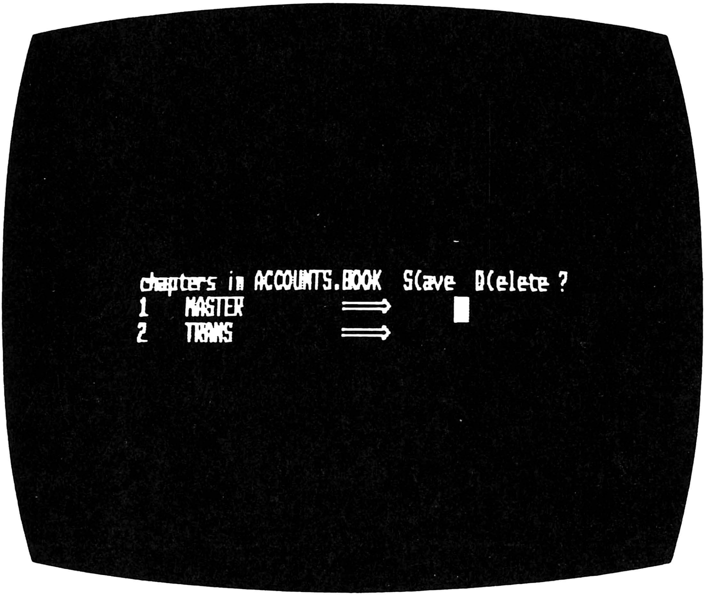
Now you are ready to enter new values for the size of the data dictionary, chapters, etc. After each of the following prompts, MICROBOOK will ask ARE YOU SURE? Enter 'Y' if you are sure, otherwise MICROBOOK will repeat the prompt.

ENTER MAX # OF KEY WORDS FOR DICTIONARY

This establishes an upper limit on the capacity of the data dictionary. If additional chapter forms are to be added later on then this limit must be increased. However, increasing this limit results in consuming more disk space. If the data dictionary is too small, the backup COP(Y command will fail.

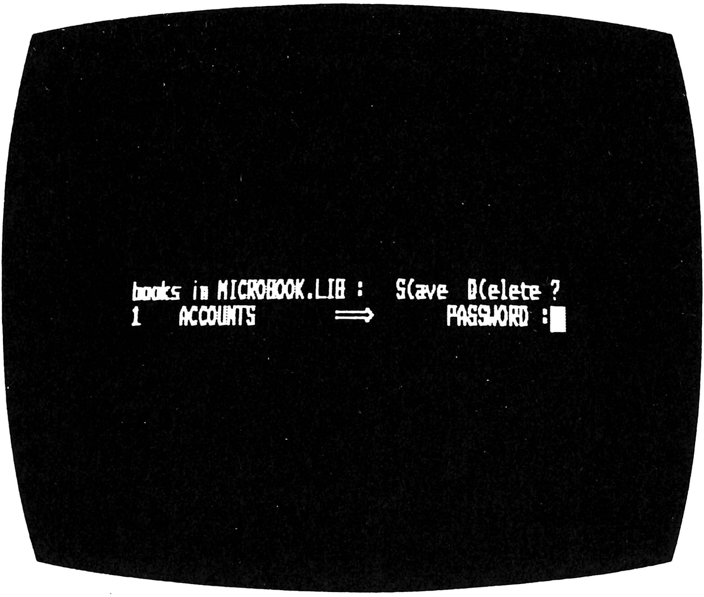
The COP(Y function selects a book to be copied, checks the password you entered, and then copies one at a time each chapter of the selected book. Each book can be saved (copied) or not (deleted). Hence, S(AVE and D(ELETE options will appear on the screen as shown in Figures 10.3, 10.4, and 10.5

If S is entered, then you must supply a correct password in no more than two trials. Failure to provide a correct password on the second trial will cause the book to be deleted (not copied).



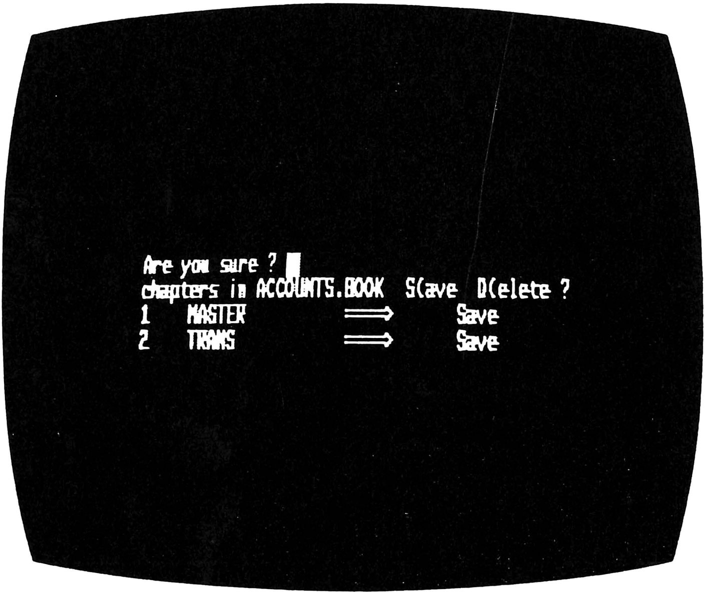
```
chapters in ACCOUNTS.BOOK  S(ave  D(elete ?
1   MASTER          ⟹      █
2   TRANS           ⟹
```

Figure 10.3 Enter S to save or else D to delete a chapter



```
books in MICROBOOK.LIB :   S(ave  D(elete ?
1    ACCOUNTS         ⟹         PASSWORD :█
```

Figure 10.4 S(aved chapters must have correct passwords before they are copied



```
Are you sure ?█
chapters in ACCOUNTS.BOOK  S(ave  D(elete ?
1    MASTER           ⟹         Save
2    TRANS            ⟹         Save
```

Figure 10.5 Enter Y if the selection is correct

Once a correct password is entered, the list of chapters is displayed on the screen along with S(AVE and D(ELETE options as shown in Figure 10.4.

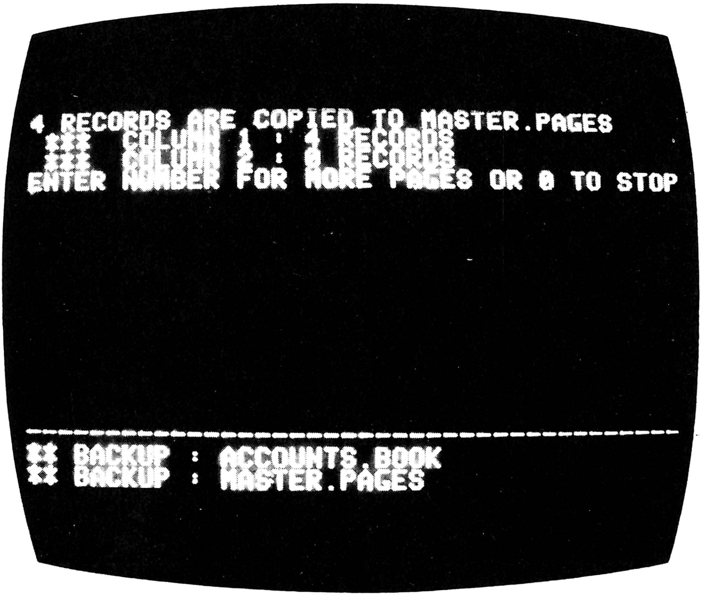
A correct chapter password must also be entered for every chapter to be saved. The pages marked DELETE (see B(ROWSE) will *not* be copied to the new disk volume.

After all non-deleted pages have been copied MICROBOOK displays the number of items copied and then requests a new limit on the maximum size of the saved chapter.

ENTER NUMBER FOR MORE PAGES OR 0 TO STOP

If you enter 0 the saved chapter is of the same length (in pages) as the original. If you enter 10, for example, the saved chapter will hold up to 10 pages more than the original chapter. See Figure 10.6.

Finally, MICROBOOK terminates each save operation (copy) with a message telling you how many pages are available in the chapter indexes which accommodate chapters containing key words. This message does not require any response.



```
4 RECORDS ARE COPIED TO MASTER.PAGES
 ***  COLUMN 1 : 4 RECORDS
 ***  COLUMN 2 : 0 RECORDS
ENTER NUMBER FOR MORE PAGES OR 0 TO STOP




---------------------------------------
** BACKUP : ACCOUNTS.BOOK
** BACKUP : MASTER.PAGES
```

Figure 10.6 You may extend the length of a chapter

Warning: The COP(Y function is sensitive to anomalies in the database. For example, you will notice unexpected EXEC errors if backing up a chapter which has *no* index words. Since every chapter must have at least one access word in order to be indexed, this should not occur. However, it does illustrate sensitivity to *unusual* database structure. As long as every chapter has at least one index file (corresponding to at least one access word), such errors will not occur.

In these unusual cases, consult the program listings for information on how to modify MICROBOOK to handle exceptions.

11

MICROBOOK Extensions

11.1 THE PURPOSE OF MICROBOOK EXTENSIONS

Many applications of the MICROBOOK data management system require processing that goes beyond the capabilities of the functions provided by the Command level and QUERY. Therefore, Pascal programs must be written which access and process MICROBOOK files. These programs are compiled and stored on the MICRO: volume. They can be executed in the same manner as QUERY.

Rather than start from scratch each time a Pascal processing program is written to extend MICROBOOK, a list of pre-compiled functions and procedures are kept in a special SYSTEM.LIBRARY file. These functions and procedures are in a unit called MICROBOOK.

The purpose of the MICROBOOK extensions, then, is to add to the processing power of MICROBOOK by adding Pascal programs which process MICROBOOK files. This is done with the aid of the MICROBOOK unit stored in SYSTEM.LIBRARY. See volume COMP:.

11.2 HOW TO USE UNIT MICROBOOK

Any Pascal program can use the pre-compiled routines of MICROBOOK by including the following statements as shown below:

```
program EXTEND ;
  uses CHAINSTUFF, MICROBOOK ;
  (* other Pascal statements *)
  SETCHAIN ('MICRO:MICROBOOK') ;
end.
```

The *uses* statement causes the two units CHAINSTUFF and MICROBOOK to be loaded with the new program. The SETCHAIN function causes control to return back to the Command level when program EXTEND terminates. Be sure to use these statements exactly as they appear above.

The details of how to use unit MICROBOOK depend on what the processing program is supposed to do. However, the specific functions included in unit MICROBOOK are listed below.

11.3 SUMMARY OF MICROBOOK ROUTINES

All data values (alphabetic and numeric) are stored in character string format. The MICROBOOK unit defines this as type STRWORD. All file names are stored as type FNAMES. Entire pages of data values are of type ARYWORD, which is

type
  ARYWORD = *array*[1..NWORDS] of STRWORD

where NWORDS is equal to the maximum number of data words per page (see C(REATE). Finally, key words are of type STRKEY which defines a string of characters equivalent to STRWORD in length.

The following summary is a guide to the use of routines stored in the MICROBOOK unit. For detailed technical explanations, see the program listings.

Arithmetic

procedure FPSTR(NUMBER : *real* ; *var* STRIN : STRWORD) ;
  Convert NUMBER into a string, STRIN.

function STRFP(STRIN : STRWORD ; *var* NUMBER:*real)* : *boolean* ;
  If STRFP is true, the string STRIN is converted into a NUMBER. If false, the conversion has failed.

procedure ADD(WORD1, WORD 2 : STRWORD ; *var* RESULT : STRWORD) ;
procedure SUB(WORD1, WORD2 : STRWORD ; *var* RESULT : STRWORD) ;
procedure MUL(WORD1, WORD2 : STRWORD ; *var* RESULT : STRWORD) ;
procedure DVD(WORD1, WORD2 : STRWORD ; *var* RESULT : STRWORD) ;

These routines perform numerical calculations on the numeric strings as follows:

WORD1 + WORD2→RESULT
WORD1 – WORD2→RESULT
WORD1 * WORD2→RESULT
WORD1 / WORD2→RESULT

procedure ZERO (*var* STRIN : STRWORD) ;

Set STRIN to a string of zeros.

Retrieval

procedure FIRST(WORD : STRWORD ; *var* KEY : STRWORD) ;

The index file defined by key word WORD is searched for the value of the first entry in order. This value is returned in KEY. If the search fails, then KEY='0'. Example: FIRST('*NAME', KEYVAL) returns KEYVAL='ADAMS'.

function LOOKUP(WORD,KEY : STRWORD) : *integer* ;

LOOKUP computes the page number of the page containing KEY in index WORD. For example: J :=LOOKUP('*NAME', 'LEWIS') computes the page number containing LEWIS. If the search fails, then LOOKUP returns null (–1).

function NEXT(WORD : STRWORD ; *var* KEY : STRWORD) : *integer*;

NEXT computes the page number of the next key in index WORD. The next key is returned to KEY, also. If the search fails then NEXT is NULL (–1) and KEY='0'. For example, to retrieve the value and page number of the next name in the *NAME index, J :=NEXT('*NAME', KEYVAL) ; This function should be used after an initial FIRST or LOOKUP routine.

function DICLOOK(WORD : STRKEY ; *var* BK, CH : FNAMES; *var* CL,LN : *integer*) : *boolean* ;

If DICLOOK is true, then WORD is in the data dictionary at the location given by the book BK, chapter CH, column number CL, and line number LN within the form of CH. If DICLOOK is false, the WORD could not be found.

procedure GETWORD(WORD : STRKEY ; *var* VALUE : STRWORD ; PAGE : *integer*) ;

procedure PUTWORD(WORD : STRKEY ; VALUE : STRWORD ; PAGE : *integer* ; XFLAG : *boolean*) ;

These procedures retrieve or store data values in pages as follows: The data dictionary is searched for WORD. The book and chapter containing WORD are retrieved and the value corresponding to WORD is retrieved from or stored in the page numbered PAGE. For example:

```
GETWORD('BAL', VAL,3) ;
PUTWORD('BAL', '0', 3, TRUE) ;
```

If the access fails a zero is returned in VAL. All book and chapter passwords must be supplied by the user at the keyboard during execution of the first access. If XFLAG is False, no indexing is performed even if the data word is a key word.

procedure PUTPAGE(WORD : STRKEY ; VALUE : ARYWORD ;
  PAGE : *integer* ; XFLAG : *boolean*) ;

procedure GETPAGE(WORD : STRKEY ; *var* VALUE :
  ARYWORD ; PAGE : *integer*) ;

These two procedures do the same thing as GETWORD and PUTWORD except they transfer an entire page of values in a single access. The values are passed through array VALUE instead of simple variable VALUE. For example, to get all data values of page #3 via any key word in the column of page #3, say *NAME:

```
GETPAGE('*NAME', VALUES, 3) ;
```

where VALUES is an array of data values. If XFLAG is False, no indexing is performed even if a key word is stored in the page. However, if XFLAG is true, all key words are inserted into their appropriate index files.

Input/Output

procedure DISPLAY(BOOK,CHAP : FNAMES ;
  COL,PAGE : *integer*) ;

This routine causes page PAGE and column COL of book BOOK and chapter CHAP to be displayed on the screen. All passwords will be requested on the first display operation to each book and chapter.

procedure INPUT(BOOK,CHAP : FNAMES) ;

The form for BOOK and CHAP will be displayed on the screen after checking passwords, etc. Then you can enter data values for one or more pages, a column at a time.

procedure PRINT(BOOK,CHAP : FNAMES ; PAGE : *integer*) ;

The form and contents of page number PAGE inside BOOK and CHAP will be printed on the line printer. If the page has two columns, they will be printed side-by-side. All passwords will be checked as needed.

***This procedure may not be implemented in unit MICROBOOK due to the differences in printers. If not, then you can implement it and include it in your program.
Note: Use LOOKUP and GETPAGE to implement PRINT so passwords will be checked.

For an example of the use of the MICROBOOK unit see the source program listing for QUERY. The QUERY language processor is implemented using the MICROBOOK unit. A short illustration follows.

11.4. AN EXAMPLE

The following example is useful only to illustrate how to use the MICROBOOK unit. We have removed the data declaration and type statements from the listing in order to keep it simple.

```
program NOTHING ;
  uses CHAINSTUFF, MICROBOOK ;
  . .
  begin
    FIRST('*NAME', MKEY) ; (* get first key *)
    if DICLOOK('*NAME',BOOK,CHAP,CL,LN) then
    begin MPAGE :=LOOKUP('*NAME', MKEY) ;
    repeat
    DISPLAY(BOOK,CHAP,CL,MPAGE) ;
    MPAGE :=NEXT('*NAME',MKEY) ;
    until MPAGE=NULL ;
    end ;
    SETCHAIN('MICRO : MICROBOOK') ;
end
```

In this example, MKEY is returned from FIRST as the first value of *NAME. Then MPAGE is the page number of the page containing MKEY. The loop displays every page that can be found until NULL is encountered. Finally, the SETCHAIN routine establishes a way back to the Command level of MICROBOOK.

12

An Example: Accounts Receivable

12.1 SETTING UP THE FORMS

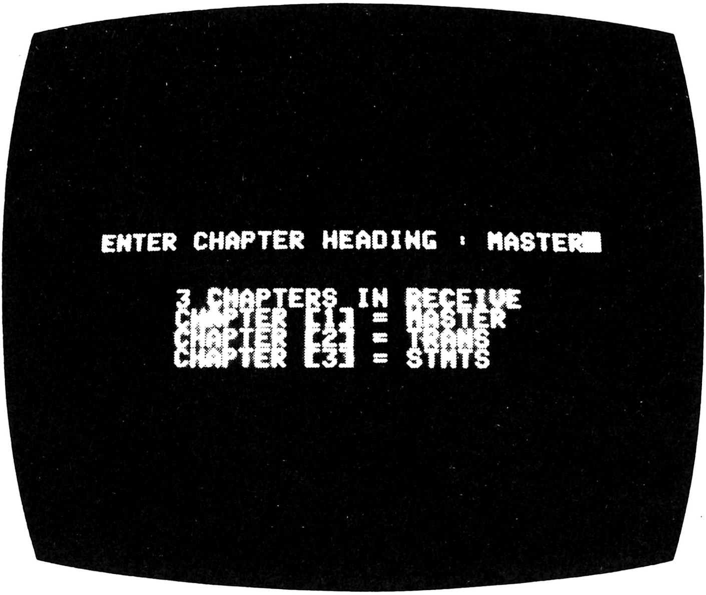
An open item accounts receivable system consists of three chapters:

MASTER: This chapter contains the name, address and current balance of every account in the accounts receivable database. Figures 12.1, 12.2, and 12.3 show how this chapter is constructed and used.

TRANS: This chapter contains the name, identification number, and amount of a single transaction for every transaction which has occurred in the current billing period. Figures 12.4, 12.5, and 12.6 show how this chapter is constructed and used. Notice that *TI is the only access key word in TRANS.

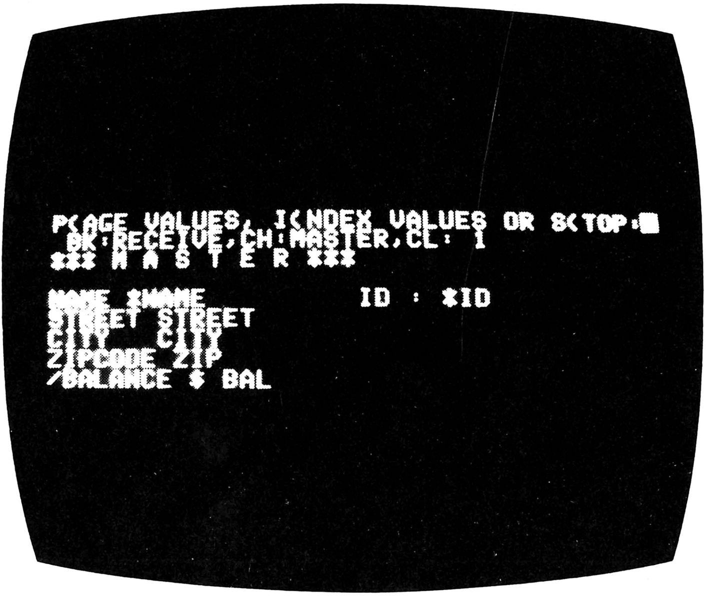
STMTS: This chapter is the *output* chapter. It consists of a name and mailing address, and a balance due field. This chapter is used to print bills to be sent to each client owing money. Figures 12.7, 12.8, and 12.9 show the construction of STMTS.

To set up the accounts receivable system, construct the book and the three chapters shown in Figures 12.1–12.9. Next, the client accounts are entered into MASTER using MICROBOOK's E(NTER function. Each MASTER page is indexed on *NAME and *ID as shown in Figure 12.2. This method of indexing was chosen so you can quickly retrieve a BALance for any client given a name or a number. Since two or more clients may possibly have the same name, a unique *ID number is assigned to each account.



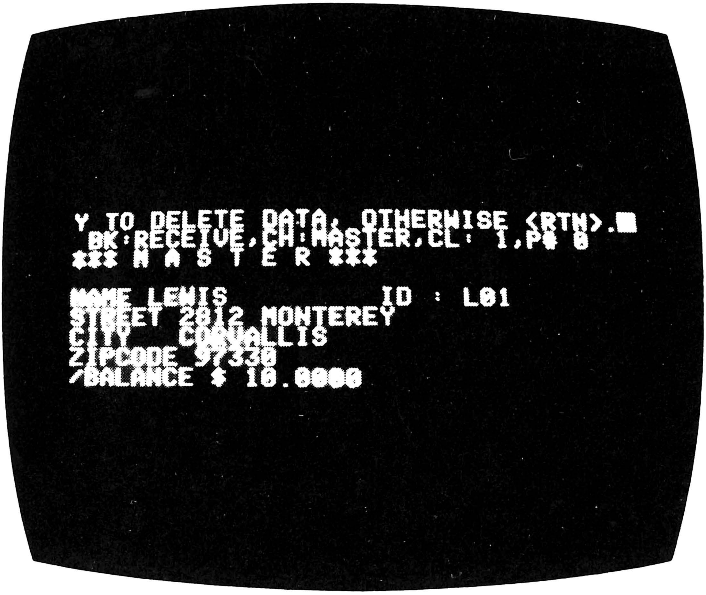
```
ENTER CHAPTER HEADING : MASTER█

     3 CHAPTERS IN RECEIVE
     CHAPTER [1] = MASTER
     CHAPTER [2] = TRANS
     CHAPTER [3] = STATS
```

Figure 12.1 Retrieve chapter MASTER



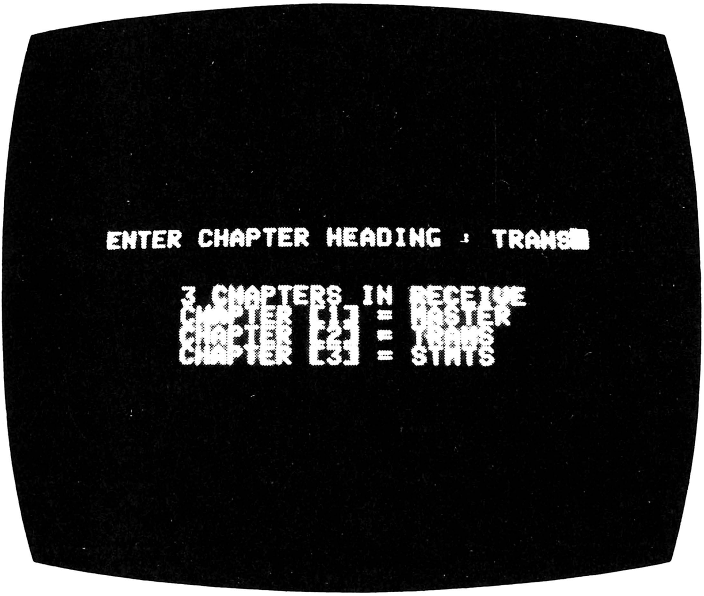
```
P(AGE VALUES, I(NDEX VALUES OR S(TOP:█
 BK:RECEIVE,CH:MASTER,CL: 1
*** M A S T E R ***

NAME $NAME             ID : *ID
STREET STREET
CITY   CITY
ZIPCODE ZIP
/BALANCE $ BAL
```

Figure 12.2 Display data words of MASTER



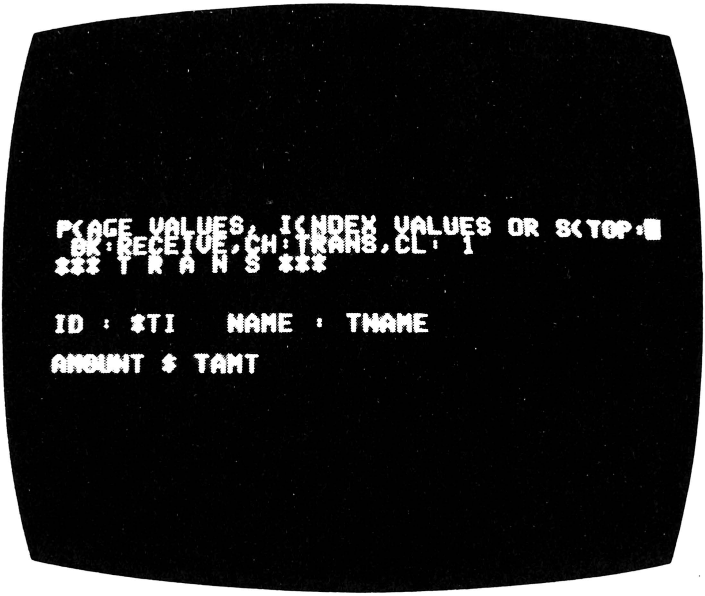
```
Y TO DELETE DATA, OTHERWISE <RTN>.■
 BK:RECEIVE,CH:MASTER,CL: 1,P# 0
*** M A S T E R ***

NAME LEWIS               ID : L01
STREET 2012 MONTEREY
CITY   CORVALLIS
ZIPCODE 97330
/BALANCE $ 10.0000
```

Figure 12.3 Display a page of data values in MASTER



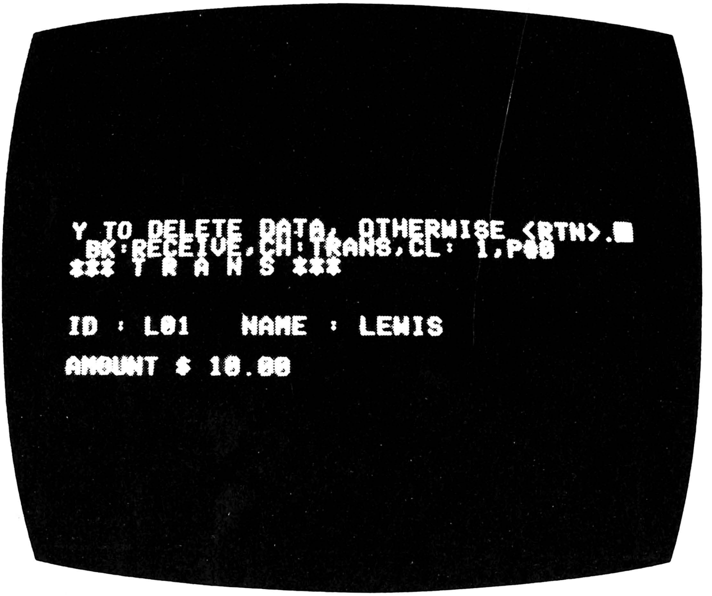
```
ENTER CHAPTER HEADING : TRANS■

      3 CHAPTERS IN RECEIVE
      CHAPTER [1] = MASTER
      CHAPTER [2] = TRANS
      CHAPTER [3] = STATS
```

Figure 12.4 Retrieve chapter TRANS



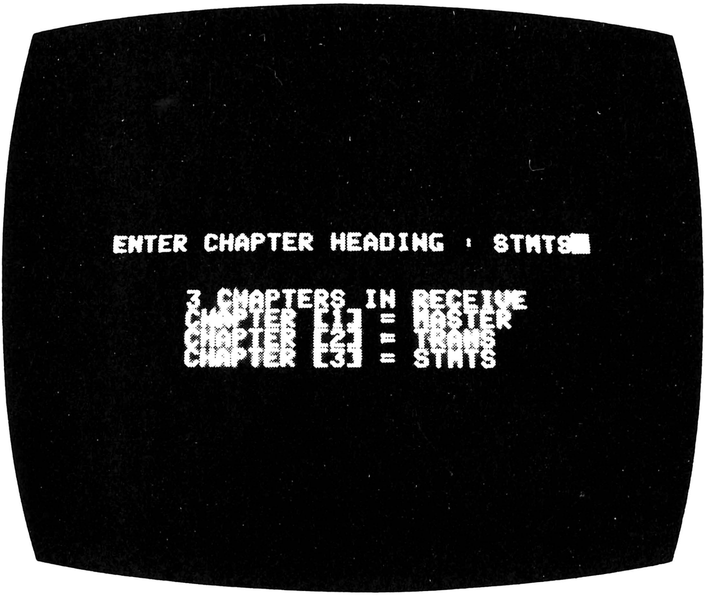
```
P<AGE VALUES, I<NDEX VALUES OR S<TOP:■
 BK:RECEIVE,CH:TRANS,CL: 1
*** T R A N S ***

ID : $TI   NAME : TNAME

AMOUNT $ TAMT
```

Figure 12.5 Display data words of TRANS



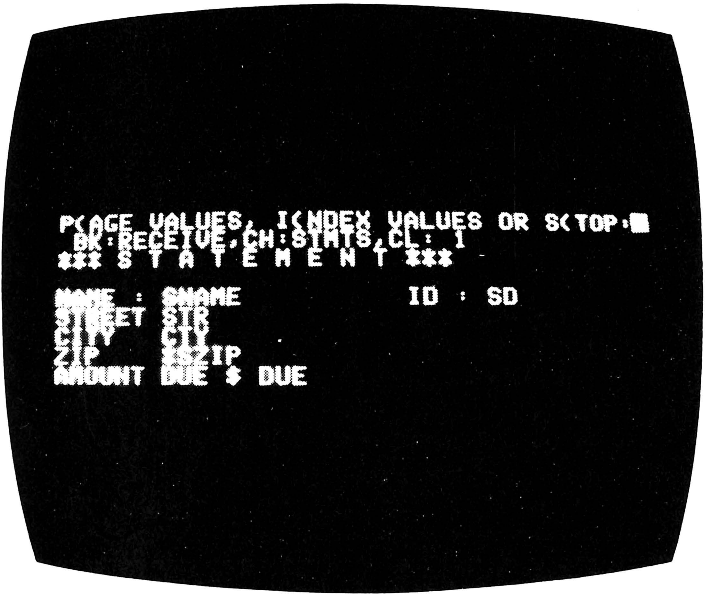
```
Y TO DELETE DATA, OTHERWISE <RTN>.■
 BK:RECEIVE,CH:TRANS,CL: 1,P00
*** T R A N S ***

ID : L01   NAME : LEWIS

AMOUNT $ 10.00
```

Figure 12.6 Display a page of data values in TRANS



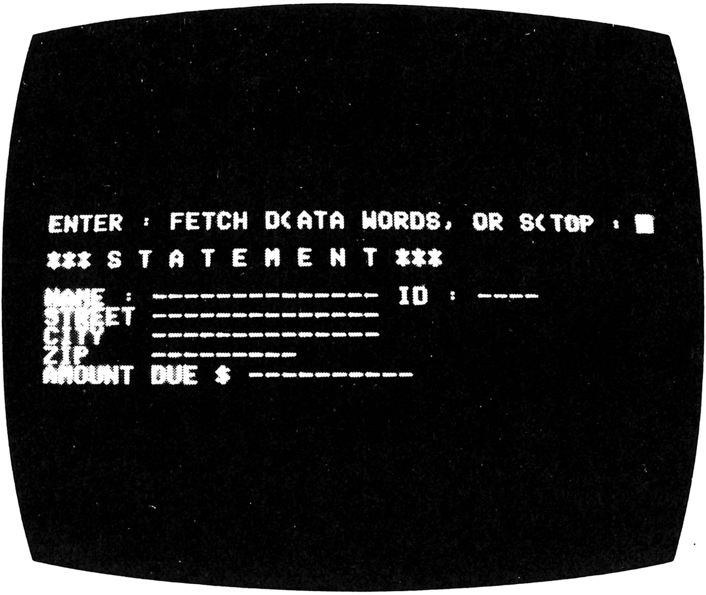
```
ENTER CHAPTER HEADING : STMTS

     3 CHAPTERS IN RECEIVE
     CHAPTER [1] = MASTER
     CHAPTER [2] = TRANS
     CHAPTER [3] = STMTS
```

Figure 12.7 Retrieve chapter STMTS



```
P(AGE VALUES, I(NDEX VALUES OR S(TOP:
 BK:RECEIVE,CH:STMTS,CL: 1
*** S T A T E M E N T ***

NAME : SNAME              ID : SD
STREET STR
CITY   CIY
ZIP    SSZIP
AMOUNT DUE $ DUE
```

Figure 12.8 Display data words of STMTS



```
ENTER : FETCH D<ATA WORDS, OR S<TOP : ■
*** S T A T E M E N T ***
NAME : ---------------- ID : ----
STREET ----------------
CITY   ----------------
ZIP    ----------
AMOUNT DUE $ -----------
```

Figure 12.9 Display an empty page of chapter STMTS

Using this accounts receivable system, you enter transactions (payments received or charges made during the billing period) one at a time and in the order they are received. Figure 12.6 shows an example of a transaction for ID = L01, and NAME = LEWIS.

Most likely, several transactions will be entered for each account. Therefore, the TRANS chapter will hold the amounts for many transactions with identical ID and NAME fields. It is the job of the posting routine to go through all transactions and collect only those that match specific MASTER accounts. You will see how this is done in the next section.

You need an output chapter to hold a trial billing list. The STMTS chapter shown in Figure 12.9 is filled-in by copying name and address information from the MASTER chapter into the STMTS chapter. Notice that this chapter is indexed by zipcode only. This method of indexing is used so that the billing statements are printed in zipcode order. The post office offers lower postage rates for bulk mail that is sorted from lowest-to-highest zipcode. When the STMTS chapter is printed, you can print each page in zipcode order using *SZIP as the access word.

The chapters of RECEIVE are organized as shown here so that you can

(1) Fetch MASTER account balances using name or identification numbers.

(2) Process MASTER and TRANS pages on unique identification numbers.

(3) Output trial billing runs to STMTS without actually printing any statements until you are sure they are correct.

(4) Keep all transactions on-line for as long as you need them – and be able to retrieve all that belong to the same account upon demand.

(5) Print the pages of STMTS in zipcode order so you can take advantage of postal rates.

Now, you are ready to process the database using the QUERY language processor.

12.2 QUERY PROCESSING

The QUERY processor is activated by EX(ECUTEing QUERY.CODE as shown in Figure 12.10. You will learn how to post and output a sample billing run in two separate query programs. However, both programs could be combined into a single query program.

Figures 12.11 and 12.12 show how to retrieve and run the POSTing program. This program sums all transactions for a given account identification number and stores the sum in BAL. Specifically,

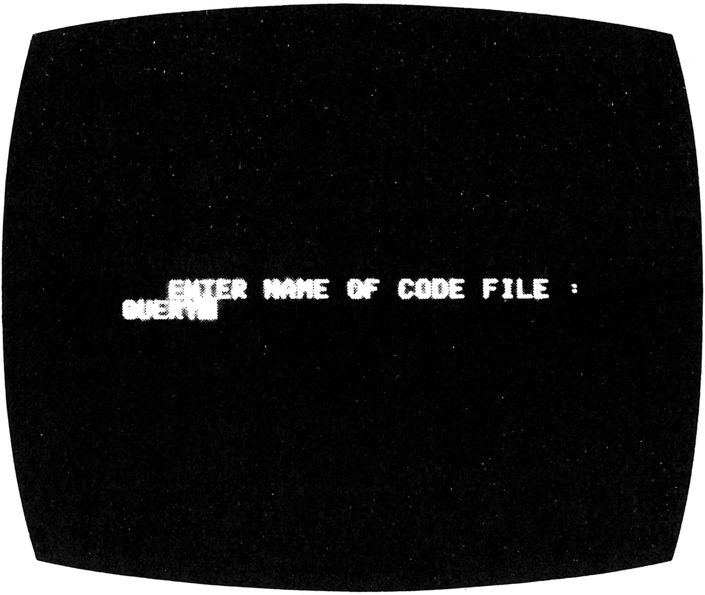
```
FOR *ID=* TO * DO ;
```

This means to retrieve all pages in the chapter containing *ID (which happens to be in chapter MASTER) from start to finish. Then,

```
LET BAL=SUM *TI=*ID TO *ID
  DO TAMT=TAMT ;
```

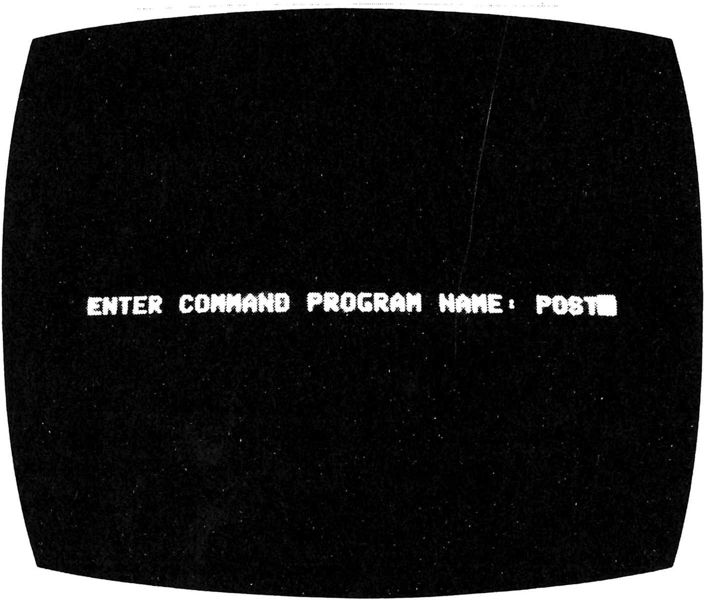
This command means to compute a value for BAL by summing all TAMT values stored in TRANS which match identification numbers for *TI equals *ID. Remember, there may be more than a single matching TRANS page since each account may have several transactions pending. Hence, the summation is computed over all of them.

When POST completes its execution, the BAL field of every MASTER account will contain zero (no transaction) or the sum



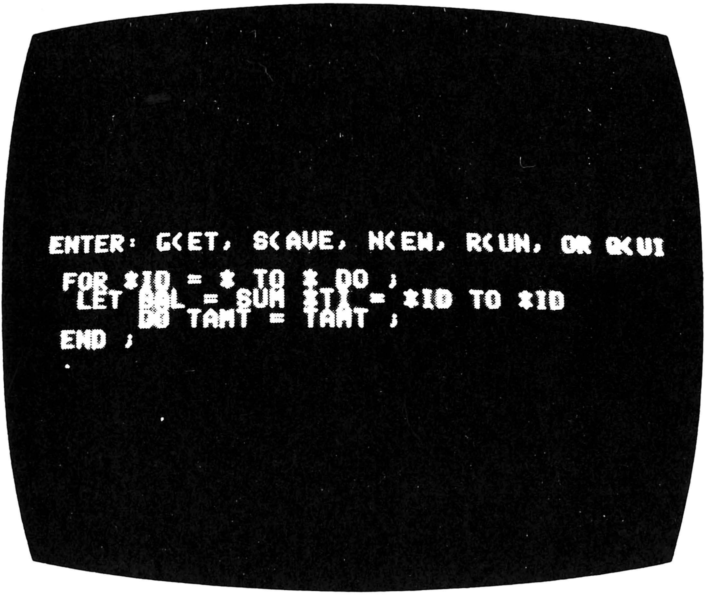
```
    ENTER NAME OF CODE FILE :
QUERY
```

Figure 12.10 EX(ECUTE QUERY language processor



```
ENTER COMMAND PROGRAM NAME: POST
```

Figure 12.11 Retrieve program POST



```
ENTER: G(ET, S(AVE, N(EW, R(UN, OR Q(UI
 FOR *ID = * TO * DO ;
  LET BAL = SUM *TI = *ID TO *ID
      DO TAMT = TAMT ;
 END ;
```

Figure 12.12 POST updates BAL in every MASTER page

of the TAMT values corresponding to its account identification number. Figure 12.3 shows a page of MASTER after POST has been run.

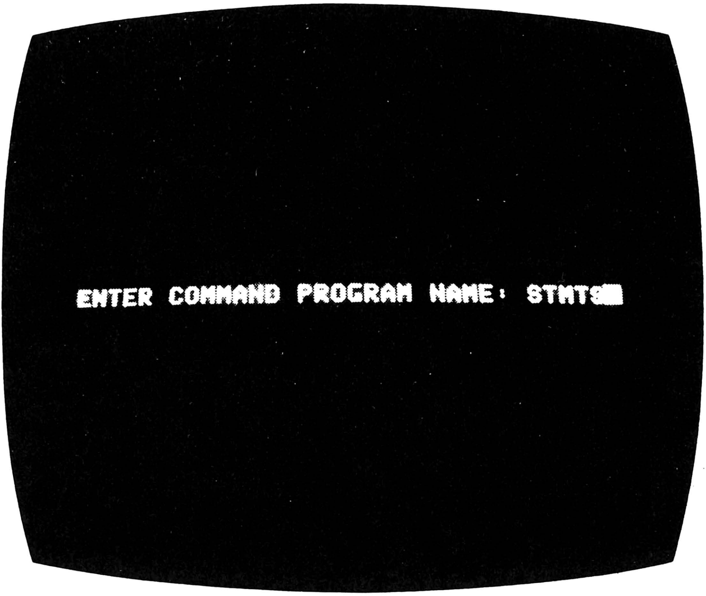
Figures 12.13 and 12.14 show the STMTS program for moving the information contained in MASTER to the pages of STMTS. Since *SZIP is an access key word, this program also re-indexes the information so that it can be accessed by zip-code number rather than name or identification number. Let's look at STMTS in more detail.

FOR *ID = * TO * DO BAL > '0' ;

This causes all pages of MASTER containing BAL greater than zero to be retrieved in ID order. All pages that do not have a value of BAL greater than zero are skipped.

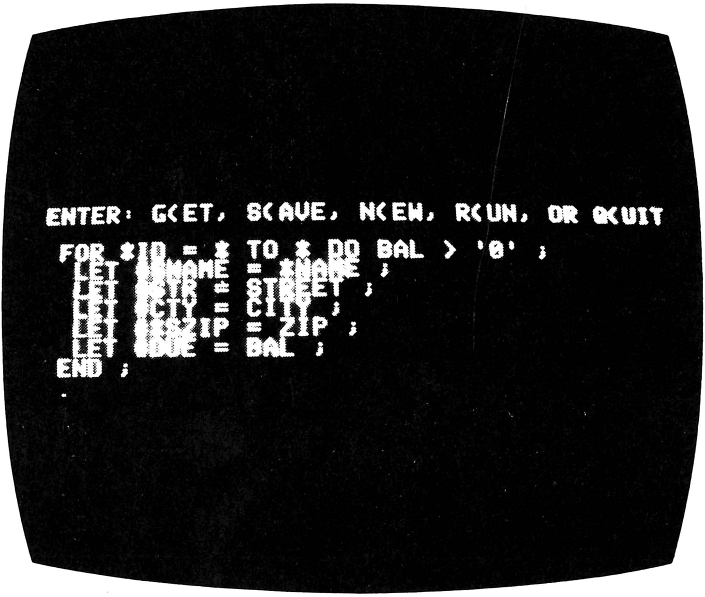
Next, we use # to indicate that a data word *does not* belong in the page retrieved by the FOR-END pair. Hence, SNAME, STR, CTY, *SZIP, and DUE are all part of chapter STMTS rather than chapter MASTER. What happens to these values?

The first pass through the FOR-END pair causes SNAME, STR, CTY, *SZIP, and DUE to be stored in page zero of chapter STMTS. The second pass causes page one to be stored; the



```
ENTER COMMAND PROGRAM NAME: STMTS█
```

Figure 12.13 Retrieve program STMTS



```
ENTER: G(ET, S(AVE, N(EW, R(UN, OR Q(UIT
 FOR #ID = $ TO $ DO BAL > '0' ;
  LET #NAME = NAME ;
  LET #STR = STREET ;
  LET #CTY = CITY ;
  LET #ZIP = ZIP ;
  LET #DUE = BAL ;
 END ;
```

Figure 12.14 Program STMTS updates chapter STMTS using #

third pass stores page two, and so forth. Figure 12.15 shows the output obtained from the second pass (page 1). Notice how ID: is not updated because the program did not copy a value from *ID to SD. How would you do this?

The statement,

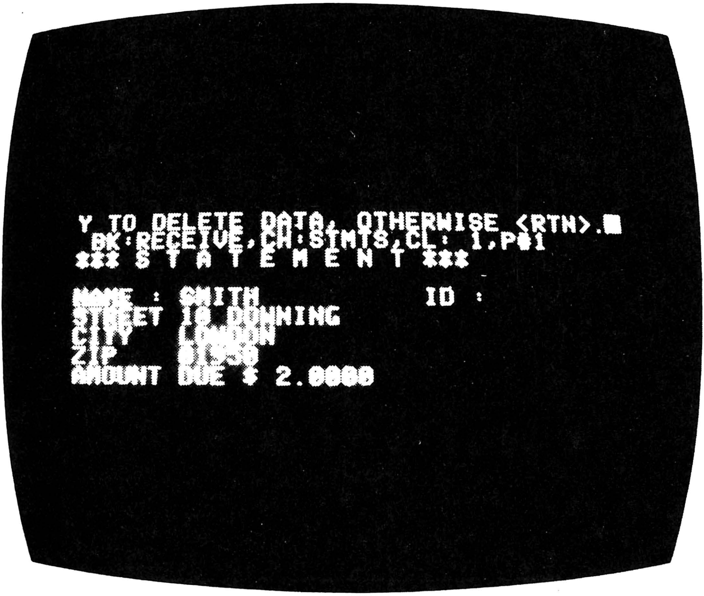
```
LET # *SZIP=ZIP ;
```

moves the value stored in ZIP (of MASTER) to the field *SZIP (in STMTS). In addition, since *SZIP is an access key word, the value of SZIP is inserted into the index file SZIP.INDEX. Now, you can access a page of STMTS by zipcode number. For example,

```
FOR *SZIP=* TO * DO ;
? ;
END ;
```

causes all pages of STMTS to be output in zipcode order.

This example demonstrates how to use MICROBOOK to solve a very common data processing problem. You can use MICROBOOK in a similar way to solve a variety of other problems by careful design of forms (chapters), and dedicated application of the QUERY language processor.



```
Y TO DELETE DATA, OTHERWISE <RTN>.■
 BK:RECEIVE,CH:STMTS,CL: 1,P01
*** S T A T E M E N T ***

NAME : SMITH               ID :
STREET 10 DOWNING
CITY   LONDON
ZIP    [illegible]
AMOUNT DUE $ 2.0000
```

Figure 12.15 Sample page of chapter STMTS after it has been updated

13

Program Listing for MICROBOOK

13.1 OVERVIEW

The source program listing for the main program of MICROBOOK is shown on the next few pages. The source text is also available from the publisher on a disk volume titled SOURCE:. This disk can be used to modify the system in a number of ways to configure MICROBOOK to new or unusual applications.

The structure of the main program is shown below in outline form:

```
program MICROBOOK
  (* global data structures *)
  (* global functions and procedures *)
  (* segments: one for each command *)
  (* main body of MICROBOOK including:
     (1) headings and command level,
     (2) case statement for selecting a segment
  *)
```

Each one of the segments is either shown in the program listing for MICROBOOK.TEXT or else shown as an include-file. The include-files are separate TEXT files which reside on the SOURCE: volume.

Additional programs also exist on the SOURCE: volume for modifying and converting the system. These are explained in a subsequent section which discussed how to modify the MICROBOOK and UNIT programs.

13.2 DATA STRUCTURE

The key to understanding how MICROBOOK actually works is held within the global data structures of MICROBOOK. The DATA.TEXT file on SOURCE: contains all data types and variables used in the database management system. A second file called QDATA.TEXT is used to keep an old copy (previous copy whenever the current copy is changed) of the data structure for the purpose of modifying then converting an old database into a modified database format. This procedure is explained in the next section.

The most important parameters in DATA.TEXT are the program constants shown in the *const* section. These are also displayed each time the C(REATE command is executed.

NBOOKS : The maximum allowable number of books which can be stored in a MICROBOOK library.

NCHAPS : The maximum allowable number of chapters which can be stored in a MICROBOOK book.

NCOLS : The maximum allowable number of columns which can be stored in a page of a chapter.

NWORDS : The maximum allowable number of fields (data words) which can be stored in a field (data word value).

MAXWORD : The maximum allowable number of characters which can be stored in a field (data word value).

NBTREE : The order or size of each node in the dictionary B-tree of MICROBOOK. This is increased to improve the performance of MICROBOOK and decreased to reduce the amount of memory required by MICROBOOK.

NPP : The order or size of each node in an index file B-tree. This is increased to improve performance, and decreased to save memory.

MAXKEY : The maximum allowable length of an access key word. This is usually identical in size with MAXWORD.

MAXCHAP : The maximum allowable length of a book/chapter name.

XORIGIN, XLIMIT, YORIGIN, YLIMIT : Defines the viewport of the screen within which forms can be displayed.

UPPER,LEFT : Defines the *home* square of the screen. The screen censor is placed here before prompting you with a system command.

NULL : This is a marker for designating the end of a file or end of an index. It should never be positive or zero.

BLANK : A blank line. This is used to erase the prompt line at location (UPPER, LEFT).

In addition, the global types listed below are accessible to any program using MICROBOOK. These types are *visible* to any program which uses the MICROBOOK unit discussed in the next chapter.

FNAMES : File name types used for books, chapters, and pages.

STRKEY : Key words are strings of this type.

STRWORD : Data words are strings of this type.

ARYWORD : A page of data words are of this type (array). This should be used to declare an array for use with GETPAGE and PUTPAGE, see the chapter on MICROBOOK extensions.

BOOKS : This is the data structure of the dictionary of MICROBOOK.

CHAPTERS : This is the data structure of each book's table of contents.

WORDS : This is the data structure for every data word in the dictionary. It contains all the information needed to locate a data word within the library of books.

PAGES : This is the data structure of a chapter form. It is used to hold any screen form defined by the user along with the data words defined by the user.

DATUM : This is the data structure of each page-column stored in a MICROBOOK page. DSTAT is FALSE if this page of data values is deleted.

NODES : This is the data structure of each node of the dictionary B-tree file.

EMPTIES : This is the data structure of the header file for the dictionary.

PPDATA : This is the string containing an access value indexed by an index B-tree file.

PLOCATION : This is the information held in each index (B-tree) file needed to locate a value somewhere within the database.

PAGEDATA : This is the data structure of each node in the index file B-tree.

PUSED : This is the data structure of the header file for an index file. NEX and NXI are used by the extended functions (FIRST, NEXT, LOOKUP) in order to find the next access value in order.

These structures are used throughout MICROBOOK. Any change in them leads to two effects which must be considered:

(1) The data structure of all other MICROBOOK routines will be altered, thus leading to a re-compilation of all source codes (including the source code in UNIT).

(2) The existing database (if any) currently stored on disk will no longer be compatible with any modified MICROBOOK program because the data structures may not be compatible.

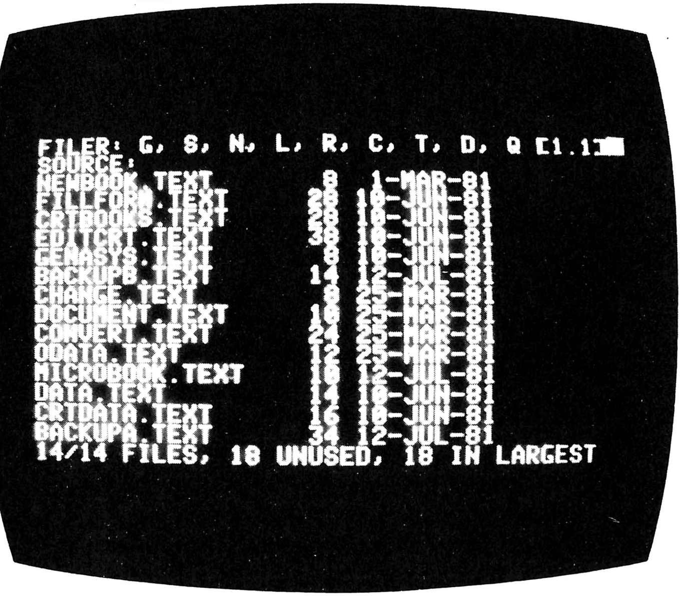
If you make one or more changes to these structures, then the MODIFY/CONVERT programs may be of use.

13.3 MODIFY/CONVERT

The program listings of MODIFY and CONVERT are included in disk volume SOURCE:. The purpose of MODIFY is to allow you to modify certain program constants without using the text editor. This program will also generate a new DATA.TEXT file containing the modified constants, and old QDATA.TEXT file containing the original data structures. These files are useful for converting an old formatted database into a current format.

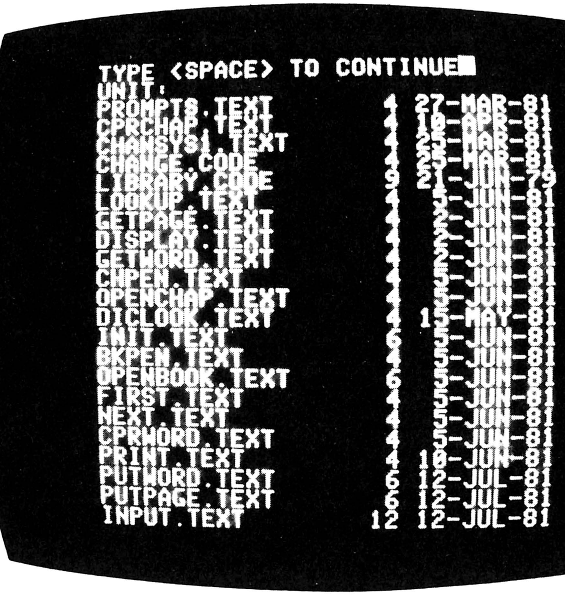
The CONVERT program takes an old formatted database stored on one volume and re-writes it onto a second volume in new DATA.TEXT format. Thus, this program can be used after MODIFY to translate databases stored in one format into a database stored in the current format. This program uses both QDATA.TEXT and DATA.TEXT.

Warning: Several limitations exist on the use of CONVERT and MODIFY. First, the DATA.TEXT and QDATA.TEXT structures must co-exist in memory when CONVERT is used. This may cause a STACK OVERFLOW if memory is too small for both. Secondly, the programs stored on volume UNIT: may be used by one or more processing programs (for example, QUERY). This means that all UNIT: programs must also be re-compiled and linked into the extended programs running under MICROBOOK. If this is the case, then the data structures of the UNIT:UNIT.TEXT file must also be modified and re-compiled, added to SYSTEM.LIBRARY, and then be used to re-compile QUERY and any other program which uses the MICROBOOK unit.



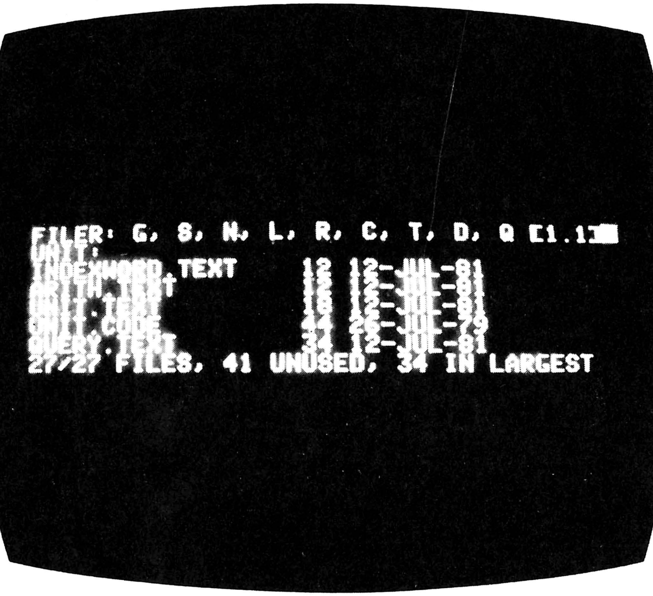
```
FILER: G, S, N, L, R, C, T, D, Q [1.1]
SOURCE:
NEWBOOK.TEXT          8  1-MAR-81
FILLFORM.TEXT        28 10-JUN-81
CRTBOOKS.TEXT        28 10-JUN-81
EDITCRT.TEXT         38 10-JUN-81
GENASYS.TEXT          8 10-JUN-81
BACKUPB.TEXT         14 12-JUL-81
CHANGE.TEXT           8 25-MAR-81
DOCUMENT.TEXT        10 25-MAR-81
CONVERT.TEXT         24 25-MAR-81
QDATA.TEXT           12 25-MAR-81
MICROBOOK.TEXT       10 12-JUL-81
DATA.TEXT            14 10-JUN-81
CRTDATA.TEXT         16 10-JUN-81
BACKUPA.TEXT         34 12-JUL-81
14/14 FILES, 10 UNUSED, 10 IN LARGEST
```

Figure 13.1 Contents of volume SOURCE:



```
TYPE <SPACE> TO CONTINUE█
UNIT:
PROMPTS.TEXT           4 27-MAR-81
CPRCHAP.TEXT           4 10-APR-81
CHANSYS1.TEXT          4 25-MAR-81
CHANGE.CODE            4 25-MAR-81
LIBRARY.CODE           9 21-JUN-79
LOOKUP.TEXT            4  5-JUN-81
GETPAGE.TEXT           4  2-JUN-81
DISPLAY.TEXT           4  2-JUN-81
GETWORD.TEXT           4  2-JUN-81
CHPEN.TEXT             4  5-JUN-81
OPENCHAP.TEXT          4  5-JUN-81
DICLOOK.TEXT           4 15-MAY-81
INIT.TEXT              6  5-JUN-81
BKPEN.TEXT             4  5-JUN-81
OPENBOOK.TEXT          6  5-JUN-81
FIRST.TEXT             4  5-JUN-81
NEXT.TEXT              4  5-JUN-81
CPRWORD.TEXT           4  5-JUN-81
PRINT.TEXT             4 10-JUN-81
PUTWORD.TEXT           6 12-JUL-81
PUTPAGE.TEXT           6 12-JUL-81
INPUT.TEXT            12 12-JUL-81
```

Figure 13.2 Contents of volume UNIT:



```
FILER: G, S, N, L, R, C, T, D, Q [1.1]█
UNIT:
INDEXWORD.TEXT      12 12-JUL-81
ARITH.TEXT          16 12-JUL-81
UNIT.TEXT           10 12-JUL-81
UNIT.CODE           24 26-JUL-79
QUERY.TEXT          34 12-JUL-81
27/27 FILES, 41 UNUSED, 34 IN LARGEST
```

Figure 13.3 Remainder of contents of volume UNIT:

13.4 LISTING

```
program MICROBOOK;
  (*$S+*)
  uses  CHAINSTUFF ;

  (*$I SOURCE: DATA*)

 procedure PROMPTS( X,Y : integer ; LINE : string ;
                    var INPUT : string ) ;
           forward ;

 function NODECOUNT ( ITEM, SLOT : integer ) : integer ;
           forward ;

 segment procedure FORMS ;
  (*$I SOURCE: EDITCRT*)
  begin
   EDITCRT ;
 end ;      (*  FORMS   *)

 segment procedure ADDBOOK ;
  (*$I SOURCE: NEWBOOK*)
  begin
   NEWBOOK ;
  end ;

 segment procedure BROWSE ;
  (*$I SOURCE: CRTBOOKS*)
  (*$I SOURCE: CRTDATA*)
 begin
  GLANCE ;
 end ;     (* BROWSE  *)

segment procedure CREATE ;
 (*$I SOURCE: GENASYS*)
 begin
  GENASYS ;
 end ;

segment procedure BACKUP ;
 (*$I SOURCE: BACKUPA*)
 (*$I SOURCE: BACKUPB*)
 begin
  COPYDB  ;
 end ;

segment procedure ENTERP ;
 (*$I SOURCE:FILLFORM*)
 begin
  FILLFORM ;
 end ;

segment procedure EXEC ;
 var NEWNAME : string;
 begin
   PAGE( OUTPUT );
   GOTOXY( LEFT+3, UPPER+7 );
   writeln('Enter name of code file : ');
   readln( NEWNAME );
   SETCHAIN( NEWNAME );
   SETCVAL ( LIBNAME ) ;
   PAGE( OUTPUT );
   WRITE( ' One moment, please.  ' );
   EXIT( MICROBOOK );
 end;


procedure PROMPTS;
var
  KEYIN   : char ;
  I       : integer ;

  begin
   repeat
    GOTOXY( X,Y ) ; WRITE( LINE ) ; READLN( INPUT ) ;
    GOTOXY( X,Y ) ; WRITE( BLANK ) ; GOTOXY( X,Y ) ;
    WRITE( 'Are you sure ?' ) ; READLN( KEYIN ) ;
   until KEYIN = 'Y' ;
  end ;  (* PROMPTS *)

 function NODECOUNT ;
 const
  ROOT = 1 ;
 var
  I, J, NODES : integer ;

 begin
  I := SLOT ;
  J := 0 ;
```

```
   repeat
    NODES := TRUNC ( ( ITEM - 1 ) / SLOT ) + 1 ;
    J := NODES + J ;
    SLOT := SLOT * I ;
   until NODES = ROOT ;
  NODECOUNT := J ;
 end ;  (* NODECOUNT *)


 begin                    (* MICROBOOK *)
  PAGE( OUTPUT )  ;
  SHORT  := COPY( BLANK, 1, MAXCHAP ) ;   (* A SHORT BLANK LINE  *)
  GOTOXY( 2, 6 ) ; WRITELN(' M I C R O B O O K' );
  WRITELN;
  WRITELN('MICROBOOK is a simulated library :' );
  WRITELN('-- Dictionary of words:' );
  WRITELN('-- Books' );
  WRITELN('-- Chapters , ' );
  WRITELN('-- Indexes.' ) ;
  WRITELN ; WRITELN ;
  WRITELN('copyright 1981.');
  GOTOXY( 0,0 ) ; WRITE( 'Enter data disk volume = ' ); READLN( PREFIX  );
  if POS( ':', PREFIX ) = 0 then PREFIX := CONCAT( PREFIX, ':' ) ;
  LIBNAME := CONCAT( PREFIX , 'MICROBOOK' ) ;
  repeat
   GOTOXY( 0,0 ) ; WRITE( BLANK ) ; GOTOXY( 0, 0 ) ;
   WRITELN(  'Commands :') ;
   WRITELN('A(dd book, B(rowse, C(reate' ) ;
   WRITELN('E(nter pages, F(orms input, ' );
   WRITELN('S(top, eX(ecute, coP(y ' ) ;
   GOTOXY( 10, 0 ) ;
   READ ( KEYIN ) ;
   case KEYIN of
    'A' : ADDBOOK;         (*  ADD NEW BOOK, INDEX, AND INITIAL  *)
    'B' : BROWSE ;         (*  BROWSE THE DATABASE -- DISPLAY IT *)
    'C' : CREATE ;         (*  SYSTEM INITIALIZATION -- STARTIT  *)
    'E' : ENTERP ;         (*  ENTER PAGES OF DATA USING A FORM  *)
    'F' : FORMS  ;         (*  ENTER PAGE FORMS FOR A CHAPTER    *)
    'S' : EXIT( PROGRAM ) ;
    'X' : EXEC   ;         (*  EXECUTE A MENU PROGRAM            *)
    'P' : BACKUP ;         (*  BACKUP THE SYSTEM -- ALL FILES -- *)
   end ;
  PAGE( OUTPUT ) ;
  until FALSE ;
end.
```

SOURCE: DATA

```
const
 NBOOKS   =002 ;                  (* MAX SIZE OF LIBRARY       *)
 NCHAPS   =003 ;                  (* MAX SIZE OF BOOK          *)
 NCOLS    =002 ;                  (* MAX NO. COLUMNS PER PAGE *)
 NWORDS   =010 ;                  (* NO. ENTRIES/PAGE          *)
 MAXWORD  =015 ;                  (* MAX NO. CHAR/DATA WORD    *)
 NBTREE   =008 ;                  (* ORDER OF B TREE           *)
 NPP      =008 ;                  (* ORDER  OF INDEXES         *)
 MAXKEY   =015 ;
 MAXCHAP  =016 ;
 XORIGIN  =001 ;  XLIMIT  =041 ;  (* SCREEN WIDTH              *)
 YORIGIN  =002 ;  YLIMIT  =010 ;  (* SCREEN LENGTH             *)
 UPPER    =000 ;  LEFT    =000 ;  (* HOME SQUARE OF CURSOR     *)
 NONE     =000 ;
 NULL     = -1 ;
 BLANK='                                                                  ';
type
 XAXIS    = XORIGIN .. XLIMIT ;   YAXIS = YORIGIN .. YLIMIT ;
 FNAMES   = string[ MAXCHAP ] ;   (* FILE NAMES ARE STRINGS *)
 STRKEY   = string [ MAXKEY]  ;   (* KEY WORDS ARE STRING   *)
 STRWORD  = string [ MAXWORD] ;   (* DATAWORDS ARE STRING   *)
 ARYWORD  = array [ 1..NWORDS ] of STRWORD ;   (* value of page  *)
 BOOKS    = packed record         (* LIST OF BOOKS IN LIBR. *)
             ITEMS  : 0..NBOOKS ;
             TITLES : array[ 1.. NBOOKS ] of FNAMES ;
             PASSWORD : array[ 1..NBOOKS ] of FNAMES ;
             BOPENED : packed array[1..NBOOKS] of boolean ;
            end ;
CHAPTERS  = packed record          (* CHAPTER HEADINGS      *)
             NCHAP  : 0..NCHAPS ;
             HEADING: array[ 1..NCHAPS ] of FNAMES ;
             PASWORD: array[ 1..NCHAPS ] of FNAMES ;
             COPENED: packed array[ 1..NCHAPS ] of boolean ;
            end ;
WORDS     = packed record          (* format of data dictionary entry *)
             WKEY   : string[ MAXKEY ] ;
             TITLE  : string[ MAXCHAP ] ;
             CHAP   : string[ MAXCHAP ] ;
             COLMN  : 0..NCOLS ;
             XPOS   : XAXIS ;
             YPOS   : YAXIS ;
             LONG   : integer ;
             PLACE  : integer ;
            end ;
PAGES      = packed record              (* FORM AND BLANK SIZES          *)
             NCOL    : 1..NCOLS ;
             MAXPAGE : integer ;    (* max. no. of pages. *)
             PROMPT  : array[ YAXIS ] of string[ XLIMIT ] ;
             NLINES  : YORIGIN..YLIMIT ;
             ENTRY   : array[ 1..NWORDS ] of WORDS ;
             NENTRIES : 1..NWORDS ;
            end ;
DATUM      = packed record              (* entry in ppg file of data     *)
             DSTAT    : boolean ;   (* TRUE=KEEP, FALSE=DELETE       *)
             DPAGENO  : integer ;   (* page number *)
             DCOL     : integer ;   (* column number *)
             VALUES   : array[ 1..NWORDS ] of STRWORD;
            end ;
NODES     = packed record               (* form of data dictionary nodes *)
            FLAG : boolean ;(* TRUE=NODE CONTAINS DATA POINTERS*)
                            (* FALSE=NODE HAS INDEX POINTERS   *)
            POINTER : array[ 1..NBTREE ] of integer  ;
            SUBKEY  : array[ 1..NBTREE ] of WORDS ;
            LINK    : integer ;
           end ;
EMPTIES   = packed record      (* header of data dictionary file *)
            TREESIZE : integer ;
            TN         : integer ;    (* NUMBER OF NODES USED    *)
            ROOT       : integer  ;
           end ;
PPDATA    = string[ MAXWORD ];  (* value stored in index file    *)
PLOCATION= packed record      (* points to value in data page  *)
            PKEY    : PPDATA ;(* data value                    *)
           PTITLE   : FNAMES ;(* book title       *)
           PHEAD    : FNAMES ;(* chapter heading  *)
           PP       : integer;(* page number      *)
           PCOLUMN  : 1..NCOLS ; (* column number *)
          end ;
PAGEDATA = packed record        (* node in index b-tree files     *)
           PFLAG    : boolean;(* T=leaf, F=internal node       *)
           PPSUBKEY: array[ 1..NPP ] of PLOCATION ;
           PPOINTER: array[ 1..NPP ] of integer ;
           PLINK    : integer ;
          end ;
PUSED    = packed record         (* list of available space of index*)
           PN       : integer;(* number of pages used, so far.  *)
           PROOT    : integer;(* current root of b-tree index   *)
           NEX      : integer;(* next pointer    *)
           NXI      : integer;(* next subscript of key   *)
          end ;

var
  LIBNAME    : STRING ;           (* name of microbook library      *)
  LIBRARY    : file of BOOKS ; (* contains the library of books *)
  CHAPHEAD   : FNAMES ;           (* chapter heading name            *)
  CONTENTS   : file of CHAPTERS;(*table of contents of a book    *)
  FORMFILE   : FNAMES ;           (* name of form file for chapter *)
  FORM       : file of PAGES ; (* file containing form of chapt *)
  BOOK       : FNAMES ;           (* working file name  for books  *)
  FILESPACE : FNAMES ;           (* name of availist file for...  *)
  AVAILIST   : file of EMPTIES;(* list of available space for...*)
  DICTION    : file of NODES ; (* index file = b-tree structure *)
  PINDEX     : file of PAGEDATA;(* index to pages of a book      *)
```

```
    AINDEX    : file of PUSED ; (* list of available space in index*)
    PPG       : file of DATUM ; (* data file containing values    *)
    KEYIN     : char ;
    LINEIN    : string ;        (* misc. prompt variables         *)
    SHORT     : string ;        (* working blank line for MAXCHAP*)
    PW        : FNAMES ;
    NPX       : INTEGER ;
    PR        : TEXT    ;       (* printer: text file              *)
    PREWORD   : STRWORD ;
    PRECHAP   : FNAMES  ;
    PREBOOK   : FNAMES  ;
    PREFIX    : FNAMES  ;       (* prefix volume name             *)


procedure EDITCRT ;

  var
    SCREEN    : array[ YAXIS ] of string[ XLIMIT ];
    LINE      : YAXIS ;
    X         : 0 .. XLIMIT ;
    Y         : 0 .. YLIMIT ;
    COL       : integer  ;          (* COLUMN COUNTER *)
    KCOL      : 1..NCOLS ;          (* NUMBER OF COLUMNS *)

    function STARTUP : boolean ;
     var
      DONE  : boolean ;
      I     : integer ;
```

```
procedure TITLED ;
 begin
  CLOSE( LIBRARY, LOCK ) ;
  RESET( LIBRARY, CONCAT( LIBNAME, '.LIB' ) ) ;
  PROMPTS( LEFT, UPPER, 'Enter book title :', LINEIN ) ;
  if LENGTH( LINEIN ) > MAXCHAP
   then BOOK := COPY( LINEIN, 1, MAXCHAP )
   else BOOK := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
  GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
  WRITE( 'Enter password : ' ) ; READLN( KEYBOARD, LINEIN ) ;
  if LENGTH( LINEIN ) > MAXCHAP
   then PW := COPY( LINEIN, 1, MAXCHAP )
   else PW := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
  STARTUP := TRUE ;
  with LIBRARY^ do
   begin
    DONE := FALSE ;
    for I := 1 to ITEMS do
     begin
      if COPY( TITLES[ I ], 1, LENGTH( BOOK ) ) = BOOK then DONE := TRUE ;
      if DONE
       then
        if PASSWORD[ I ] <> PW
           then
            begin
             PROMPTS( LEFT, UPPER, 'Password wrong. Strike <return>.',
                      LINEIN ) ;
             DONE := FALSE ;
            end ;  (* if *)
     end ;  (*  for loop  *)
     if not DONE
      then
       begin
         CLOSE( LIBRARY, LOCK ) ;
         STARTUP := FALSE ;
         PROMPTS( LEFT, UPPER,
         'Error in library. Strike <return>.', LINEIN ) ;
         EXIT( STARTUP ) ; (* BOOK NOT FOUND IN LIB *)
       end ;
     CLOSE( DICTION, LOCK ) ; CLOSE( AVAILIST, LOCK ) ;
     RESET( DICTION, CONCAT( LIBNAME, '.DICT' ) ) ;
     RESET( AVAILIST, CONCAT( LIBNAME, '.AVAIL' ) ) ;
    end ;  (*  with  *)
   CLOSE( LIBRARY, LOCK ) ;
   CLOSE( CONTENTS, LOCK ) ;
   RESET( CONTENTS, CONCAT( PREFIX, BOOK, '.BOOK' ) ) ;
   GOTOXY( 5, 3 ) ;
   WRITE( CONTENTS^.NCHAP, ' Chapters in ', BOOK ) ;
   for I := 1 to CONTENTS^.NCHAP do
    begin
     GOTOXY( 5, 3+I ) ;
     WRITE( 'Chapter [', I, '] = ', CONTENTS^.HEADING[ I ] ) ;
    end ;
   if CONTENTS^.NCHAP = NCHAPS
    then begin
     WRITELN('Book is full.' ) ;
     CLOSE( CONTENTS, LOCK ) ;
     EXIT( PROGRAM );
    end ;
end ;    (*  titled   *)
```

```
begin
   STARTUP := FALSE ;
   TITLED ;
   PROMPTS( LEFT, UPPER, 'Enter chapter heading : ', LINEIN ) ;
   if LENGTH( LINEIN ) > MAXCHAP
    then CHAPHEAD := COPY( LINEIN, 1, MAXCHAP )
    else CHAPHEAD := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
   GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
   WRITE( 'Enter password : ' ) ; READLN( KEYBOARD, LINEIN ) ;
   if LENGTH( LINEIN ) > MAXCHAP
    then PW := COPY( LINEIN, 1, MAXCHAP )
    else PW := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
   with CONTENTS^ do
    begin
     DONE := TRUE ;
     for I := 1 to NCHAP  do
       if CHAPHEAD = HEADING[ I ]
         then
          begin
           DONE := FALSE ;
           NCHAP := I - 1 ;    (* JUST IN CASE REPLACE IT *)
          end ;
     if not DONE
      then
       begin
        PROMPTS( LEFT, UPPER,
        'Chapter exists. R(eplace, Q(uit ?', LINEIN ) ;
        if COPY( LINEIN, 1, 1 ) = 'Q'
         then
          begin
```

```
           CLOSE ( CONTENTS, LOCK ) ;
           STARTUP := FALSE ;
           EXIT( STARTUP ) ;
          end ;   (* if COPY *)
        end ;   (* if not DONE  *)
      NCHAP := NCHAP + 1 ;     (* ADD TO CONTENTS  *)
      HEADING[ NCHAP ] := CHAPHEAD ;
      PASWORD [ NCHAP ] := PW ;
     end ;  (* with *)
    SEEK( CONTENTS, 0 ) ;
    PUT( CONTENTS ) ;
    CLOSE( CONTENTS, LOCK ) ;
    REWRITE( FORM, CONCAT( PREFIX, CHAPHEAD, '.FORM' ) ) ;
  end ;    (* STARTUP  *)

 procedure FINISH ;
   begin
     SEEK ( AVAILIST, 0 ) ;
     PUT ( AVAILIST ) ;
     CLOSE( AVAILIST, LOCK ) ;
     CLOSE( DICTION, LOCK ) ;
   end ;                         (* FINISH *)

function  INDEXDATA( DATAWORD : WORDS  ) : boolean;
(*$G+ *)
  label
    4 ;

  type
    KEYS      =  WORDS ;         (* SEARCH  KEYS                 *)
    POINTERS =  integer ;        (* FILE POINTERS IN B TREE      *)
```

```
var
  I, K, LAST : integer ;      (* WORKING COUNTERS, ETC             *)
  NEW, TEMP  : NODES   ;      (* WORKING NODES                     *)
  NEWNODE,
  NODE,
  TEMPPTR    : POINTERS ;     (* WORKING POINTERS                  *)
  STOP       : -1 .. 20 ;     (* STACK POINTER                     *)
  STACK      : array[ 0..20 ] of POINTERS ;
  KEY        : KEYS ;         (* THE SEARCH KEY                    *)
  ITEM       : POINTERS ;     (* THE DATA RECORD POINTER TO SFILE*)

  function ALOCATE : integer ;
    begin
      with AVAILIST^ do
        begin
              TN := TN + 1 ;
              if TN > TREESIZE then
               begin
                GOTOXY( LEFT, UPPER ) ;
                WRITE( 'Out of data dictionary space. ',
                       'Strike <return>.' ) ;
                READ ( KEYIN ) ;
                EXIT( PROGRAM ) ;
               end
              else  ALOCATE := TN ;
         end ;
  end;                        (*  ALOCATE    *)

    function  LOOKUP : boolean ;
      begin
  LOOKUP := TRUE ;
  STOP := 0 ; STACK[ STOP ] := NULL ;
  STOP := 1 ; STACK[ STOP ] := NULL ;
  NODE := AVAILIST^.ROOT ;       (* START AT ROOT NODE        *)
  repeat
    I := 1 ;
    SEEK( DICTION, NODE ) ;
    GET ( DICTION ) ;
    with DICTION^ do
      begin
        while ( SUBKEY[ I ].WKEY <> COPY( BLANK, 1, MAXKEY )  )
                    and
              ( SUBKEY[ I ].WKEY < KEY.WKEY )
         do
          begin
          I := SUCC( I ) ;         (* FIND INSERT PLACE           *)
          end ;
        if SUBKEY[ I ].WKEY = KEY.WKEY
           then
            begin
             LOOKUP := FALSE ;
             EXIT ( LOOKUP ) ;
            end ;
        STOP := STOP + 1 ; STACK[ STOP ] := NODE ;
        STOP := STOP + 1 ; STACK[ STOP ] := I ;
        NODE := POINTER[ I ] ;
      end ;                          (* with *)
  until ( NODE = NULL ) or ( DICTION^.FLAG );
end ;                                (*  LOOKUP  *)
```

```
procedure SPLIT ;
  begin
    with  DICTION^ do
      begin
        NEW.LINK := LINK ;
        NEW.FLAG := FLAG ;
        LAST     := ( NBTREE  + 1) div 2 ;
        K        := 1 ;
        while LAST+K <=  NBTREE  do
          begin
            NEW.POINTER[ K ] := POINTER[ LAST + K ] ;
            NEW.SUBKEY [ K ] := SUBKEY [ LAST + K ] ;
            K := SUCC ( K ) ;
          end ;              (* while *)
        NEW.POINTER[ K ] := TEMPPTR ;
        NEW.SUBKEY [ K ].WKEY := COPY( BLANK, 1, MAXKEY ) ;
        for K := K+1 to  NBTREE  do
          begin
            NEW.POINTER[ K ] := NULL ;
            NEW.SUBKEY [ K ].WKEY := COPY( BLANK, 1, MAXKEY ) ;
          end ;              (* while *)
      end ;                (* with  *)

   (*   STEP   7   *)

   TEMP    := DICTION^ ;
   DICTION^  := NEW ;
   NEWNODE := ALOCATE ;
   SEEK( DICTION, NEWNODE ) ;
   PUT ( DICTION ) ;
```

```
         DICTION^  := TEMP ;
         with DICTION^ do
           begin
             KEY  := SUBKEY[ LAST ] ;
             for K := LAST+1  to  NBTREE  do
               begin
                 SUBKEY [ K ].WKEY := COPY( BLANK, 1, MAXKEY ) ;
                 POINTER[ K ] := NULL ;
               end ;                       (*  for  *)
             LINK := NEWNODE ;
           end ;               (* with *)
         SEEK( DICTION, NODE ) ;
         PUT ( DICTION ) ;
    end ;                (* SPLIT *)

        (******    BODY OF INDEXDATA     *****)

begin
   INDEXDATA := TRUE ;
   KEY := DATAWORD ;
   ITEM := NULL ;
   if not LOOKUP
     then
      begin
       INDEXDATA := FALSE ;
       EXIT ( INDEXDATA ) ;
      end ;
   I         := STACK[ STOP ] ; STOP := STOP - 1 ;
   NODE      := STACK[ STOP ] ; STOP := STOP - 1 ;
   NEW.FLAG := TRUE ;
```

```
(*      STEP   4        *)

4:
 with DICTION^ do
   begin
     TEMPPTR := POINTER[  NBTREE   ] ;
     for K := NBTREE downto I + 1 do
       begin
         POINTER[ K ] := POINTER[ K - 1 ] ;
         SUBKEY [ K ] := SUBKEY [ K - 1 ] ;
       end ;
     POINTER[ I ] := ITEM ;
     SUBKEY [ I ] := KEY ;
     FLAG         := NEW.FLAG ;
   end ;            (*  with  *)

   (*   STEP   5    *)

   if DICTION^.SUBKEY[  NBTREE   ].WKEY = COPY( BLANK, 1, MAXKEY )
     then
       begin
         SEEK( DICTION, NODE ) ;
         PUT ( DICTION ) ;
         INDEXDATA := TRUE  ;
         EXIT( INDEXDATA ) ;
       end ;

   SPLIT ;

   ITEM    := NODE ;
   I       := STACK[ STOP ] ; STOP := STOP - 1 ;
```

```
         NODE   := STACK[ STOP ] ; STOP := STOP - 1 ;
         if NODE = NULL
           then
             begin
               with  DICTION^ do
                 begin
                   FLAG := FALSE ;
                   POINTER[ 1 ] := ITEM ;
                   SUBKEY [ 1 ] := KEY ;
                   POINTER[ 2 ] := NEWNODE ;
                   SUBKEY [ 2 ].WKEY := COPY( BLANK, 1, MAXKEY );
                   for K := 3 to  NBTREE  do
                     begin
                       POINTER[ K ] := NULL ;
                       SUBKEY [ K ].WKEY := COPY( BLANK, 1, MAXKEY ) ;
                     end ;
                   LINK := NULL ;
                   NEWNODE := ALOCATE ;
                 end ;          (* with *)
                 SEEK( DICTION, NEWNODE ) ;
                 PUT ( DICTION ) ;
                 AVAILIST^.ROOT := NEWNODE ;
                 EXIT( INDEXDATA );
               end ;     (* IF-THEN  *)
      SEEK( DICTION, NODE ) ;
      GET ( DICTION ) ;
      DICTION^.POINTER[ I ] := NEWNODE ;
      NEW.FLAG := FALSE ;
      goto 4 ;

end;                    (* INDEXDATA *)
```

```
function  FORMSIZE : integer;
 var
   COUNT, I, START : integer ;
   STR             : string ;

 begin
   I := 0 ;
   for Y := YORIGIN to YLIMIT do
     begin
       STR := CONCAT( SCREEN[ Y ], ’.’ ) ;
       X := 0 ;                      (* REMEMBER XPOS    *)
       while ( LENGTH( STR ) > 1 ) and ( POS( ’-’, STR ) > 0 ) do
         begin
           I := I + 1 ;                   (* COUNT ENTRIES IN FORM  *)
           COUNT := 0 ; START := POS( ’-’, STR ) ;
           X := X + START    ;
           while COPY( STR, START+COUNT, 1 ) = ’-’ do
                 COUNT := COUNT + 1 ;
           FORM^.ENTRY[ I ].XPOS := X-1 ;
           X := X + COUNT - 1 ;              (* LOCATE NEXT FIELD  *)
           FORM^.ENTRY[ I ].YPOS := Y ;
           FORM^.ENTRY[ I ].LONG := COUNT ;
           FORM^.ENTRY[ I ].PLACE:= I  ;
           STR := COPY( STR, START+COUNT, LENGTH( STR )-START-COUNT+1 );
         end ;  (* while *)

     end ;  (* FOR *)
     FORMSIZE := I ;         (* RETURN MAX NO ENTRIES  *)
 end ;      (* FORMSIZE *)
```

```
procedure DICTIONARY ;

var
  X         : 0 .. XLIMIT ;
  Y         : 0 .. YLIMIT ;
  N , M     : 0 .. NWORDS ;
  SIZE      : 0 .. XLIMIT ;
  DONE      : boolean ;               (* end-of-LINE SWITCH          *)
  NOPAGE, NOREC, I, TEMP : integer ;

procedure GETFIELD( XX, YY, SIZE, N : integer ) ;
  var
    TEMP    : string[ XLIMIT ] ;
  begin
    with FORM^.ENTRY[ N ] do
      begin
         GOTOXY ( XX, YY ) ;
         TITLE := BOOK ;
         CHAP  := CHAPHEAD ;
         COLMN := COL ;
         PLACE := N ;
         XPOS  := XX ;
         YPOS  := YY ;
         LONG  := SIZE ;       (* REMEMBER WHERE WORD  IS       *)
         READLN( TEMP );
         if LENGTH( TEMP ) > LONG
           then  WKEY  := COPY( TEMP, 1, LONG )
           else  WKEY  := TEMP ;       (* TRUNCATE if TOO LONG FOR FLD *)
        end ;                           (*      with ENTRY              *)
   end ;                              (*      GETFIELD               *)
```

```
procedure PUTWORDS ;

procedure MAKEFILES( XWORD : string  ) ;
 var
  J  : integer ;
  SHORT : string[ MAXKEY ] ;
 begin
  REWRITE( PINDEX, CONCAT( PREFIX, XWORD, '.INDEX' ) ) ;
  with PINDEX^ do
   begin
    PFLAG := FALSE ;
    SHORT := COPY( BLANK, 1, MAXKEY ) ;
    for J := 1 to NPP do
     begin
      PPSUBKEY[ J ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
      PPSUBKEY[ J ].PTITLE := SHORT ;
      PPSUBKEY[ J ].PHEAD := SHORT ;
      PPSUBKEY[ J ].PP := 0 ;
      PPSUBKEY[ J ].PCOLUMN := 1 ;
      PPOINTER[ J ] := NULL ;
     end ;  (* for J  *)
    PLINK := NULL ;
   end ;  (* with PINDEX  *)
   for J := 1 to NPX do
    begin
     SEEK( PINDEX, J-1 ) ;
     PUT( PINDEX ) ;
    end ;
   GOTOXY ( LEFT, UPPER ) ;WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
   WRITE( XWORD, '.INDEX: page index size=',
          NPP * NPX, ' entries.' ) ;
 CLOSE( PINDEX, LOCK ) ;

 REWRITE( AINDEX, CONCAT( PREFIX, XWORD, '.AVAIL' ) ) ;
 SEEK( AINDEX, 0 ) ;
 AINDEX^.PN := 0 ;  AINDEX^.PROOT := 0 ;
 PUT( AINDEX ) ;
 CLOSE ( AINDEX,LOCK ) ;
 GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
 WRITE( XWORD, '.AVAIL: available space =',
        NPX, ' for index ' ) ;
end ;  (*  MAKEFILES  *)

begin          (*  PUTWORDS  *)
 with FORM^ do
  begin
   for M := 1 to N do
    begin
     repeat
      DONE := INDEXDATA( ENTRY[ M ] ) ;
      if not DONE
       then
        begin
         GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ;
         GOTOXY( LEFT, UPPER ) ; WRITE( 'Word ', ENTRY[ M ].WKEY,
         ' exists. Change it. OK?' ) ;
         READ ( KEYIN ) ;
         PROMPTS( LEFT, UPPER,'Change word to : ', LINEIN ) ;
         if LENGTH( LINEIN ) > MAXKEY
           then ENTRY[ M ].WKEY := COPY( LINEIN, 1, MAXKEY )
           else ENTRY[ M ].WKEY := LINEIN ;
      end ;
```

```
        until DONE ;
       GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ;
       GOTOXY( LEFT, UPPER ) ; WRITE( ENTRY[ M ].WKEY,' inserted .' ) ;
       GOTOXY( ENTRY[ M ].XPOS, ENTRY[ M ].YPOS ) ;
       WRITE ( COPY( BLANK, 1, ENTRY[ M ].LONG ) ) ;
       if POS( '*', ENTRY[ M ].WKEY ) = 1
         then MAKEFILES( COPY( ENTRY[ M ].WKEY, 2,
          LENGTH( ENTRY[ M ].WKEY)-1 ) );
      end ;    (*  for M  *)
     end ;  (*  with FORM^  *)
   SEEK( FORM, COL - 1 ) ;
   PUT ( FORM ) ;
  end ;    (*  PUTWORDS *)


begin     (*   D I C T I O N A R Y   *)
        RESET( FORM, CONCAT( PREFIX, CHAPHEAD, '.FORM' ) ) ;
        GOTOXY (LEFT, UPPER) ;
        WRITE ('Enter maximum no. of pages : ') ;
        READLN (NOPAGE) ;
        for COL := 1 to FORM^.NCOL do
         begin
          SEEK (FORM, COL-1) ;
          GET  (FORM) ;
          FORM^.MAXPAGE := NOPAGE ;
          SEEK (FORM, COL-1) ;
          PUT  (FORM) ;
         end ;
        REWRITE (PPG, CONCAT (PREFIX, CHAPHEAD, '.PAGES') ) ;
        with PPG^ do
  begin
   DSTAT := TRUE ;
   DPAGENO := NULL ;
   DCOL    := YORIGIN ;
   for I := 1 to NWORDS do
       begin
        VALUES[ I ] := COPY (BLANK, 1, MAXWORD ) ;
       end ;
  end ;
NOREC := NOPAGE*FORM^.NCOL + FORM^.NCOL - 1 ;
for I := 0 to NOREC do
   begin
    SEEK ( PPG, I ) ;
    PUT  (PPG) ;
   end ;
CLOSE (PPG, LOCK) ;
WRITE ('Chapter  ', CHAPHEAD, ' inserted. Strike <rtn>.') ;
READ (KEYIN) ;
TEMP := TRUNC (( NPP - 1 ) / 2) + 1 ;
NPX := NODECOUNT (FORM^.MAXPAGE, TEMP  ) ;
COL := 1 ;
SEEK (FORM, COL - 1) ;
GET (FORM) ;
repeat
 PAGE( OUTPUT ) ;
 GOTOXY ( LEFT, UPPER ) ; WRITE(
 'Type <rtn> to enter names, col=',COL:2 ) ;
 READ ( KEYIN ) ;
 with FORM^ do
  begin
```

```
GOTOXY (LEFT, YORIGIN) ;
for Y := YORIGIN to NLINES do WRITELN ( PROMPT[ Y ] ) ;
N  := 0 ;
for Y := YORIGIN to YLIMIT do
 begin
  X := POS( '-',PROMPT[ Y ] ) ;
   while      (    X > 0      )
                  AND
              ( X<=LENGTH( PROMPT[ Y ] ) )
    do
     begin
       SIZE := 0 ;
       DONE := FALSE;
       while ( ( X+SIZE ) <= LENGTH( PROMPT[ Y ] ) )
                           AND
                       ( NOT DONE )
         do
           if PROMPT[ Y ] [ X+SIZE ] = '-'
             then  SIZE := SIZE + 1
             else  DONE := TRUE ;     (* end of while-if-then-else *)

       if SIZE > 0
         then
           begin
             N     := N + 1 ;
             GETFIELD( X- 1 , Y , SIZE , N ) ;
             X     := X + SIZE ;
           end
         else X    := X + 1 ;
      end ;                               (*         while                  *)
```

```
            end ;                                  (*       for Y                *)
         (*   make any changes you want to  *)
      repeat
        GOTOXY( LEFT, UPPER ) ;  WRITE( BLANK ) ;
        GOTOXY( LEFT, UPPER ) ;  WRITE(
        'Enter 0(stop, or #(word to change:' ) ;
        READLN ( M ) ;                          (* CHANGE WORD  #M            *)
        if( M > 0 ) and ( M <= N )
          then
           with FORM^.ENTRY[ M ] do
              GETFIELD( XPOS, YPOS, LONG, M ) ;
      until M <= 0 ;
      (***** INSERT EACH ENTRY INTO DICTION, ONE AT A TIME  *****)
      PUTWORDS ;
      COL := COL + 1 ;
    end ;  (* with *)
   if COL <= FORM^.NCOL then GET( FORM ) ;
  until COL > FORM^.NCOL ;
  CLOSE ( FORM, LOCK ) ;
end;      (*  DICTIONARY  *)


 procedure CORRECT ;
  begin
   repeat
      GOTOXY ( LEFT, UPPER ) ;           (* ANY CORRECTIONS ??          *)
      WRITE  ( 'Enter 0(stop, or #(line to change :' ) ;
      READLN ( LINE ) ;
      if LINE > 0
        then
```

```
          begin
            GOTOXY ( XORIGIN, LINE + 1 ) ;  (* MOVE CURSOR to BAD LINE    *)
            WRITELN ( BLANK ) ; GOTOXY( XORIGIN, LINE +1 ) ;
            READLN ( SCREEN[ LINE + 1 ] ) ; (* REPLACE IT WITH NEW LINE   *)
            GOTOXY ( LEFT, UPPER ) ;    (* MOVE CURSOR to HOME        *)
          end ;                         (* END IF-THEN CLAUSE         *)
     until LINE <= 0 ;

 end ;    (*  CORRECT  *)




begin      (* M A I N     B O D Y *)
    PAGE( OUTPUT ) ;
    if not STARTUP then EXIT( EDITCRT ) ;

    repeat
     GOTOXY ( LEFT, UPPER ) ;           (* MOVE CURSOR to TOP OF PROMPT*)
     WRITELN( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
     WRITE( 'Enter number of columns ( 1..',NCOLS:2,') : ' ); READLN( KCOL ) ;
    until ( KCOL > 0 ) and ( KCOL <= NCOLS ) ;

    for COL := 1 to KCOL do
      begin
       PAGE( OUTPUT ) ;GOTOXY( LEFT, UPPER ) ;
       WRITELN( 'Enter screen form for column [', COL:2, ' ] :' ) ;
       GOTOXY( LEFT, UPPER+1 ) ;
 WRITE( 'Max. lines = ', YLIMIT, ' Max. words = ', NWORDS ) ;
 GOTOXY( LEFT, YLIMIT ) ; WRITELN( 'Must enter down to here.' );
 GOTOXY ( XORIGIN, YORIGIN ) ;       (* START AT THE BEGINNING     *)

 for Y := YORIGIN to YLIMIT - 1 do   (* READ EACH LINE, ONE A TIME *)
   begin
     READLN( SCREEN [ Y ] ) ;
     GOTOXY ( XORIGIN, Y + 1 ) ;     (* MOVE DOWN A LINE           *)
   end ;
 WRITE  ( 'Finish up' ) ;            (* WARNING to USER            *)

 GOTOXY ( XORIGIN, YLIMIT ) ;        (* OVERWRITE WARNING LINE     *)
 READLN ( SCREEN[ YLIMIT ] ) ;       (* READ LAST LINE IN SCREEN   *)
 CORRECT ;
            (*  WRITE CORRECTED FORM to FORMFILE  *)

 GOTOXY ( LEFT, UPPER ) ; WRITELN ( BLANK ) ;
 GOTOXY ( LEFT, UPPER ) ; WRITE   ( 'Saving column[',COL:2,' ]' ) ;

 with FORM^ do
  begin

   for Y := YORIGIN to YLIMIT do
     PROMPT[ Y ] := SCREEN[ Y ] ;(* COPY SCREEN LINE to FILE *)

   NCOL := KCOL ;               (* NUMBER OF COLUMNS IN CHAP *)
   NLINES := YLIMIT ;
   NENTRIES := FORMSIZE ;        (* COMPUTE SIZE OF ENTRIES   *)
 end ;  (* with *)
SEEK( FORM, COL - 1 ) ;
```

```
      PUT( FORM ) ;
     end ; (* for COL := 1 to KCOL *)

    CLOSE  ( FORM, LOCK ) ;              (* MAKE FILE PERMANENT       *)
    GOTOXY ( LEFT, UPPER ) ;
    FORMFILE := CONCAT( CHAPHEAD, '.FORM' ) ;
    WRITE ( 'Form in ', FORMFILE, ' . Strike <rtn>.'  ) ;
    READ( KEYIN ) ;
    PAGE (OUTPUT);
    DICTIONARY ;
    FINISH ;
 end;   (*   EDITCRT    *)


procedure NEWBOOK  ;
 var
  J,I : integer ;

  procedure FINI ;
   begin
     PAGE( OUTPUT ) ;
     REWRITE( CONTENTS, CONCAT( PREFIX, BOOK, '.BOOK' ) ) ;
     CONTENTS^.NCHAP := 0 ;
     for J := 1 to NCHAPS do
       begin
       CONTENTS^.PASWORD[ J ] := SHORT ;
       CONTENTS^.HEADING[ J ] := SHORT ;
       end ;
     SEEK( CONTENTS, 0 ) ;
     PUT( CONTENTS ) ;
    CLOSE ( CONTENTS, LOCK ) ;
    GOTOXY( UPPER, LEFT ); WRITE( BLANK ) ; GOTOXY( UPPER, LEFT ) ;
    WRITE ( 'Book ', BOOK,'[',I:2,']', ' added to library.' ) ;
    SEEK( LIBRARY, 0 ) ;
    PUT( LIBRARY ) ;
    CLOSE( LIBRARY, LOCK ) ;
    GOTOXY( 0, 1 ) ; WRITE( ' Strike <return> ' );
    READ ( KEYIN ) ;
  end ;     (** fini   **)



begin
  PAGE ( OUTPUT ) ;
  RESET( LIBRARY,CONCAT( LIBNAME, '.LIB' )  ) ;
  with LIBRARY^ do
   begin
     GOTOXY( 1, 2 ) ; WRITELN('Current books in ',LIBNAME,'.LIB : ' ) ;
     for J:=1 to NBOOKS do
       begin
        GOTOXY( 1, 3+J ) ;
        WRITELN('TITLE[',J:2,'] = ',TITLES[J],'.' );
       end ;
     if ITEMS + 1 > NBOOKS
       then
        begin
         WRITE( 'Library is full. Strike <rtn>.' ) ;
         READ ( KEYIN ) ;
         PAGE ( OUTPUT ) ;
        end
```

```
else
 begin
  PROMPTS( LEFT, UPPER, 'Enter title of book :', LINEIN  ) ;
  if LENGTH( LINEIN ) > MAXCHAP
    then BOOK := COPY( LINEIN, 1, MAXCHAP )
    else BOOK := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
  GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
  WRITE( 'Enter password :' ) ; READLN( KEYBOARD, LINEIN ) ;
  if LENGTH( LINEIN ) > MAXCHAP
    then PW := COPY( LINEIN, 1, MAXCHAP )
    else PW := COPY( LINEIN, 1, LENGTH( LINEIN ) ) ;
  TITLES[ ITEMS+1 ] := BOOK ;
  PASSWORD[ ITEMS+1 ] := PW ;
  I := 1 ;
  while TITLES[ I ] <> BOOK do I := I + 1 ;

  if I < ITEMS + 1
    then
     begin
      GOTOXY ( LEFT, UPPER ) ;
      WRITE (
      'Book exists. Enter Q(uit, R(eplace?' ) ;
      READ ( KEYIN ) ;
        case KEYIN of
         'Q' : begin
                CLOSE ( LIBRARY, LOCK ) ;
                EXIT( NEWBOOK )  ;
               end ;
         'R' : begin
                PASSWORD[ ITEMS + 1 ] := COPY( BLANK, 1, MAXCHAP ) ;
```

```
                        TITLES[ ITEMS + 1 ] := COPY( BLANK, 1, MAXCHAP ) ;
                        PASSWORD[ I ] := PW ;
                        TITLES[ I ] := BOOK ;
                       end ;
                end ;
           end  (* then clause *)
          else  ITEMS := ITEMS + 1 ;
        end ;     (* if ITEMS+1 else *)
     end ;        (*  with    *)
    FINI ;
 end ;          (*  NEWBOOK        *)


procedure GLANCE ;
  var
    IT, ITEM : integer ;
    FOUND    : boolean ;


function BANNER ( TOP : string ) : char ;
  begin
    GOTOXY (LEFT, UPPER); WRITE (BLANK) ;
    GOTOXY (LEFT, UPPER); WRITE (TOP);
    READ (KEYIN);
    BANNER := KEYIN;
  end ;  (* BANNER *)

 procedure SETPOSITION ;
  begin
   GOTOXY (XORIGIN, XORIGIN) ;
  WRITE (BLANK) ;
  GOTOXY (XORIGIN, XORIGIN) ;
 end ;    (* SETPOSITION *)

procedure WRFORM ;
 var
    I : 1..YLIMIT  ;

 begin    (* WRFORM *)
  GOTOXY (LEFT, YORIGIN) ;
  with FORM^ do
   for I := YORIGIN to NLINES do
        WRITELN (PROMPT[ I ]) ;
 end ;      (* WRFORM *)

procedure WRDATA ;
 var
    I : 1..NWORDS ;
    begin
          with FORM^ do
           for I := 1 to NENTRIES do
            begin
             GOTOXY (ENTRY[ I ].XPOS, ENTRY[ I ].YPOS) ;
             WRITE ( COPY (BLANK, 1, ENTRY[ I ].LONG) ) ;
             GOTOXY (ENTRY[ I ].XPOS, ENTRY[ I ].YPOS) ;
             WRITE (PPG^.VALUES[ I ]) ;
            end ;  (* for *)
      end ;   (*  WRDATA  *)
```

```
procedure SHOWFORM (CHAPHD : FNAMES; CL : integer ) ;
  var
    I : integer ;

  function SEARCH (IDXVAL : PPDATA; var WRD : integer ) : boolean ;
   var
      I, NODE : integer ;
  begin
      NODE := AINDEX^.PROOT ;
      SEARCH := FALSE ;
      repeat
       I := 1 ;
       SEEK ( PINDEX, NODE) ; GET (PINDEX) ;
       with PINDEX^ do
        begin
         while (PPSUBKEY[ I ].PKEY <> COPY (BLANK, 1, MAXWORD) )
         and   (PPSUBKEY[ I ].PKEY < IDXVAL) do I := SUCC( I ) ;
         if (PPSUBKEY[ I ].PKEY = IDXVAL) and (PFLAG)
            then
             begin
              WRD := I ;
              SEARCH := TRUE ;
             end ;     (* if *)
         NODE := PPOINTER[ I ] ;
        end ;     (* with *)
      until (NODE = NULL) or (PINDEX^.PFLAG) ;
  end ;   (* SEARCH *)

  procedure DELDAT ;
   var
```

```
    I : integer ;
 begin
  KEYIN := BANNER ('Y to delete data, otherwise <rtn>.') ;
  if KEYIN = 'Y'
     then for I := 1 to FORM^.NCOL do
           begin
            SEEK (PPG, ( PPG^.DPAGENO * FORM^.NCOL + I -1 ) ) ;
            GET  (PPG) ;
            PPG^.DSTAT := FALSE ;
            SEEK (PPG, ( PPG^.DPAGENO * FORM^.NCOL + I - 1) ) ;
            PUT  (PPG) ;
            PAGE ( OUTPUT ) ;
            SETPOSITION ;
            WRITE ('Data of column ', I:2, ' deleted. <rtn>.') ;
            READ (KEYIN) ;
            SETPOSITION ;
           end ;
end ;    (* DELDAT *)

procedure IDEX ;
 var
    NIDEX : 0..NWORDS ;
    AIDEX : array[ 1..NWORDS ] of string[ MAXKEY ] ;
    IDEXNM : string[ MAXKEY ] ;
    WD : integer ;
    FOUND : boolean ;
    I : integer ;

 function NOIDEX : boolean ;
  var
```

```
   I : integer ;
begin
 NIDEX := 0 ;
 NOIDEX := TRUE ;
 with FORM^ do
  for I := 1 to NENTRIES do
   if POS ('*', ENTRY[ I ].WKEY) = 1
      then
       begin
        NOIDEX := FALSE ;
        NIDEX  := NIDEX + 1 ;
        AIDEX[ NIDEX ] := COPY (ENTRY[ I ].WKEY, 2,
                                LENGTH (ENTRY[ I ].WKEY) - 1) ;
       end ;
end;  (* NOIDEX *)

procedure DISIDX ;
 var
   NODE, I : integer ;
   DAT : PPDATA ;

 procedure NEXTIDX (var WORD : integer ; var IDEXVAL : PPDATA) ;
  begin
   WORD := WORD + 1 ;
   with PINDEX^ do
     if PPSUBKEY[ WORD ].PKEY = COPY (BLANK, 1, MAXWORD)
      then if PLINK = NULL
       then
        begin
         PAGE (OUTPUT) ;
     SETPOSITION ;
     WRITE ('No more index values. Strike <rtn>') ;
     READ (KEYIN) ;
     EXIT (NEXTIDX) ;
    end
   else
    begin
     SEEK (PINDEX, PLINK) ;
     GET (PINDEX) ;
     WORD := 1 ;
    end ;
SEEK (PPG, ( PINDEX^.PPSUBKEY[ WORD ].PP*FORM^.NCOL+ CL - 1) ) ;
GET  (PPG) ;
if PPG^.DSTAT = TRUE
          then
           begin
            WRFORM ;
            SETPOSITION ;
            WRITELN ('Bk:', PINDEX^.PPSUBKEY[ WORD ].PTITLE ,
                     ',ch:', PINDEX^.PPSUBKEY[ WORD ].PHEAD,
                     ',cl:', CL:2, ',p#',
                      PINDEX^.PPSUBKEY[ WORD ].PP) ;
            WRDATA ;
            DELDAT ;
           end
          else
           begin
            SETPOSITION ;
            WRITE ('Data has been deleted. <rtn>.') ;
            READ (KEYIN) ;
```

```
               SETPOSITION ;
              end ;
 end ;  (* NEXTIDX *)

procedure ANYIDX (var WORD : integer ; var IDEXVAL : PPDATA) ;
 begin
     WRFORM ;
     SETPOSITION ;
     GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ; GOTOXY (LEFT, UPPER) ;
     WRITE ('Enter index value : ') ;
     READLN (IDEXVAL) ;
     if not SEARCH (IDEXVAL, WORD)
        then
         begin
          SETPOSITION ;
          WRITE (IDEXVAL, ' not found. <rtn>.') ;
          READ (KEYIN) ;
          SETPOSITION ;
         end     (* if not FOUND *)
        else
         begin
          SEEK (PPG, ( PINDEX^.PPSUBKEY[ WORD ].PP*FORM^.NCOL
                     + CL - 1) ) ;
          GET  (PPG) ;
          if PPG^.DSTAT = TRUE
             then
              begin
               SETPOSITION ;
               WRITELN ('Bk:', PINDEX^.PPSUBKEY[ WORD ].PTITLE ,
                        ',ch:', PINDEX^.PPSUBKEY[ WORD ].PHEAD,
                        ',cl:', CL:2, ',p#',
                        PINDEX^.PPSUBKEY[ WORD ].PP) ;
               WRDATA ;
               DELDAT ;
              end
             else
              begin
               SETPOSITION ;
               WRITE ('Data has been deleted. <rtn>.') ;
               READ (KEYIN) ;
               SETPOSITION ;
              end ;
         end ;    (* else *)
 end ;  (* ANYIDX *)

begin
 NODE := AINDEX^.PROOT ;
 SEEK (PINDEX, NODE ) ;
 GET (PINDEX) ;
 I := 1 ;
 with PINDEX^ do
  begin
   if (not PFLAG) and (PPOINTER[ I ] = NULL)
    then
     begin
      SETPOSITION ;
      WRITE ('No value in ', IDEXNM, ' index file. <rtn>') ;
      READ (KEYIN) ;
      SETPOSITION ;
      CLOSE (PINDEX, LOCK) ;
      CLOSE (AINDEX, LOCK) ;
      EXIT (IDEX) ;
     end
    else
     while not PFLAG do
      begin
       NODE := PPOINTER[ I ] ;
       SEEK (PINDEX, NODE) ;
       GET (PINDEX) ;
      end ;
   RESET (PPG, CONCAT ( PREFIX, PPSUBKEY[ I ].PHEAD, ’.PAGES’ ) ) ;
   SETPOSITION ;
   WRITE (PPSUBKEY[ I ].PKEY, ’ is 1st index value in ’, IDEXNM) ;
  end ;  (* with *)
 WD := 1 ;
 repeat
  case BANNER (’I(ndex val.,N(ext index val.,S(top:’)
  of
   ’I’ : ANYIDX (WD, DAT) ;
   ’N’ : NEXTIDX (WD, DAT) ;
   ’S’ : begin
          CLOSE (PPG, LOCK) ;
          EXIT (DISIDX) ;
         end ;
  end ;  (* case *)
 until FALSE ;
 CLOSE (PPG, LOCK) ;
end ;   (* DISIDX *)
```

```
begin   (* IDEX *)
 if NOIDEX
    then
     begin
      SETPOSITION ;
      WRITE ('No index file in column ', CL:2, ' <rtn>.') ;
      READ (KEYIN) ;
      SETPOSITION ;
      EXIT (IDEX) ;
     end ;
 KEYIN := 'N' ;
 repeat
  if KEYIN = 'N'
     then
      begin
       SETPOSITION ;
       WRITE ('Index file(s) : ') ;
       for I := 1 to NIDEX do
        WRITE (COPY (AIDEX[ I ], 1, LENGTH(AIDEX[ I ]) ), ' ' ) ;
       WRFORM ;
       repeat
        FOUND := FALSE ;
        GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ; GOTOXY (LEFT, UPPER) ;
        WRITE ('Enter index file name : ') ;
        READLN (IDEXNM) ;
        for I := 1 to NIDEX do
         if IDEXNM = AIDEX[ I ] then FOUND := TRUE ;
       until FOUND ;
       RESET (PINDEX, CONCAT( PREFIX, IDEXNM, '.INDEX') ) ;
       RESET (AINDEX, CONCAT( PREFIX, IDEXNM, '.AVAIL') ) ;
```

```
        DISIDX ;
       end ;     (* if KEYIN = 'N' *)
    repeat
     KEYIN := BANNER ('Enter  N(ew index file or S(top : ') ;
    until (KEYIN = 'S') or (KEYIN = 'N') ;
    CLOSE (PINDEX, LOCK) ;
    CLOSE (AINDEX, LOCK) ;
  until KEYIN = 'S' ;
  PAGE (OUTPUT) ;
 end ;    (* IDEX *)

procedure PGFILL ;
 var
    M : integer ;
    PNO : integer ;
 begin       (* PGFILL *)
  RESET (PPG, CONCAT( PREFIX, CHAPHD, '.PAGES') ) ;
  PNO := PPG^.DPAGENO ;
  while (not EOF (PPG) ) and (PPG^.DPAGENO <> NULL) do
   begin
    PNO := PPG^.DPAGENO ;
    GET (PPG) ;
   end ;
  if PNO = NULL
     then
      begin
       SETPOSITION ;
       WRITE ('No data values in the chapter. <rtn>.') ;
       READ (KEYIN) ;
       SETPOSITION ;
     CLOSE (PPG, LOCK) ;
     EXIT (PGFILL) ;
    end ;    (* if PNO = NULL *)
repeat
 repeat
  GOTOXY( LEFT, UPPER+1 ) ;
  WRITE( BLANK ) ; GOTOXY( LEFT, UPPER+1 ) ;
  WRITE( 'Max. page # = ', PNO:4 ) ;
  GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ; GOTOXY (LEFT, UPPER) ;
  WRITE ('Page # ( # < 0 to stop ) = ' ) ;
  READLN (M) ;
 until M <= PNO ;
 if M >= 0
    then with FORM^ do
          begin
           PAGE (OUTPUT) ;
           WRFORM ;
           SEEK (PPG, ( M*NCOL + CL - 1 ) ) ;
           GET  (PPG) ;
           if PPG^.DSTAT = TRUE
              then
               begin
                WRDATA ;
                SETPOSITION ;
                WRITELN ('Bk:', COPY (BOOK, 1, LENGTH(BOOK) ),
                         ',ch:', COPY (CHAPHD, 1, LENGTH(CHAPHD) ),
                         ',cl:', CL:2, ',p#', M:2) ;
                DELDAT ;
               end      (* then with *)
              else
```

```
                 begin
                  SETPOSITION ;
                  WRITE ('Data has been deleted. <rtn>.') ;
                  READ (KEYIN) ;
                  SETPOSITION ;
                 end ;    (* else *)
             end ;
   until M < 0 ;
   PAGE (OUTPUT) ;
   CLOSE (PPG, LOCK) ;
  end ;     (* PGFILL *)

procedure DICTFILL ;
 var
    N : 1..NWORDS ;
 begin   (* DICTFILL *)
  with FORM^ do
   begin
    for N := 1 to NENTRIES do
     begin
      GOTOXY (ENTRY[ N ].XPOS, ENTRY[ N ].YPOS) ;
      WRITE (COPY (BLANK, 1, ENTRY[ N ].LONG) ) ;
      GOTOXY (ENTRY[ N ].XPOS, ENTRY[ N ].YPOS) ;
      WRITE  (ENTRY[ N ].WKEY) ;
     end ;   (* for *)
    SETPOSITION ;
    WRITELN ('Bk:', COPY (BOOK, 1, LENGTH(BOOK) ),
             ',ch:', COPY (CHAPHD, 1, LENGTH(CHAPHD) ),
             ',cl:', CL:2) ;
   end ;     (* with *)
```

```
    repeat
     case BANNER ('P(age values, I(ndex values or S(top:') of
      'P' : PGFILL ;
      'I' : IDEX ;
      'S' : EXIT (DICTFILL) ;
     end ;
    until FALSE ;
   end ;    (*  DICTFILL *)

  begin       (* SHOWFORM *)
   SEEK (FORM, CL-1) ;
   GET (FORM) ;
   PAGE (OUTPUT) ;
   WRFORM ;
   case BANNER ('Enter : Fetch D(ata words, or S(top : ') of
         'D' : DICTFILL ;
         'S' : EXIT (SHOWFORM) ;
   end ;     (* end case *)
  end ;   (* SHOWFORM *)


procedure CRTBOOKS ;

  procedure CRTFORMS ;
   var
      COL : 1..NCOLS ;
      I, J : integer;
   begin   (* CRTFORMS *)
    PROMPTS (LEFT, UPPER, 'Enter chapter heading : ', LINEIN) ;
    CHAPHEAD := LINEIN ;
```

```
GOTOXY (LEFT, UPPER) ;
WRITE ('Enter chapter password : ') ;
READLN (KEYBOARD, LINEIN) ;
PW := LINEIN ;
with CONTENTS^ do
 begin
  FOUND := FALSE ;
  for ITEM := 1 to NCHAP do
    if HEADING[ ITEM ] = CHAPHEAD
       then begin FOUND := TRUE ; IT := ITEM end ;
    if (not FOUND) or (PASWORD[ IT ] <> PW)
       then
        begin
         CLOSE (CONTENTS, LOCK) ;
         GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ; GOTOXY (LEFT, UPPER) ;
         if FOUND
            then WRITE ('Wrong chapter password.  <rtn>.')
            else WRITE ('Chapter not found. <rtn>.');
         READ (KEYIN) ;
         EXIT (CRTFORMS) ;
        end ;    (* end if *)
   end ;         (* end with *)
  CLOSE (CONTENTS, LOCK) ;
  RESET (FORM, CONCAT ( PREFIX, CHAPHEAD, '.FORM') ) ;
  repeat
   case BANNER ('Enter : F(orms display, or S(top : ') of
    'F' : begin
           repeat
            GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ;
            GOTOXY (LEFT, UPPER) ;
```

```
              WRITE ('Enter col. no. of page form 1..',FORM^.NCOL:2, ' : ') ;
              READLN (COL) ;
             until (COL > 0) and (COL <= FORM^.NCOL) ;
             SHOWFORM (CHAPHEAD, COL) ;
            end ;   (* end 'F' *)
       'S' : begin
              PAGE (OUTPUT) ;
              CLOSE (FORM, LOCK) ;
              EXIT (CRTFORMS) ;
             end ;
      end ;   (* end case *)
     until FALSE ;
     PAGE (OUTPUT) ;
     CLOSE (FORM, LOCK) ;
 end ;   (* CRTFORMS *)


begin    (* CRTBOOKS *)
  RESET (LIBRARY, CONCAT (LIBNAME, '.LIB') ) ;
  GOTOXY (5, 3) ;
  WRITE (LIBRARY^.ITEMS, ' Books in ', LIBNAME) ;
  if LIBRARY^.ITEMS = 0
     then
      begin
       GOTOXY (40,3) ;
       WRITE ('Strike <return>.') ;
       CLOSE (LIBRARY, LOCK) ;
       READ (KEYIN) ;
       EXIT ( CRTBOOKS ) ;
      end ;
 for ITEM := 1 to LIBRARY^.ITEMS do
  begin
   GOTOXY (5, 3+ITEM) ;
   WRITE ('Book [', ITEM, '] = ', LIBRARY^.TITLES[ ITEM ]) ;
  end ;
repeat
 case BANNER ('Enter : O)pen book, or S)top : ') of
  'O' : begin
         GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ;
         GOTOXY (LEFT, UPPER) ;
         PROMPTS (LEFT, UPPER, 'Enter title of book : ', LINEIN) ;
         BOOK := LINEIN ;
         GOTOXY (LEFT, UPPER) ;
         WRITE ('Enter book password : ') ;
         READLN (KEYBOARD, LINEIN) ;
         PW := LINEIN ;
         with LIBRARY^ do
          begin
           FOUND := FALSE ; IT := 1 ;
           for ITEM := 1 to ITEMS do
               if TITLES[ ITEM ] = BOOK
                  then begin FOUND := TRUE; IT := ITEM end ;
           if (PASSWORD[ IT ] <> PW) or (not FOUND)
            then
             begin
              CLOSE (LIBRARY, LOCK) ;
              GOTOXY (LEFT, UPPER); WRITE (BLANK);
              GOTOXY (LEFT, UPPER);
              if FOUND
                 then WRITE ('Wrong book password. <rtn>.')
              else WRITE ('Book not found. <rtn>.') ;
           READ (KEYIN) ;
           EXIT (CRTBOOKS) ;
          end ;   (* end if *)
        end ;     (* end with *)
       CLOSE (LIBRARY, LOCK);
       RESET (CONTENTS, CONCAT ( PREFIX, BOOK, '.BOOK') ) ;
       PAGE (OUTPUT) ;
       GOTOXY (5, 3) ;
       WRITE (CONTENTS^.NCHAP, ' Chapters in ', BOOK) ;
       if CONTENTS^.NCHAP <> 0
          then
           begin
            for ITEM := 1 to CONTENTS^.NCHAP do
             begin
              GOTOXY (5, 3+ITEM) ;
              WRITE ('Chapter [', ITEM, '] = ', CONTENTS^.HEADING[ ITEM ]) ;
             end ;
            CRTFORMS ;
           end
          else
           begin
            CLOSE (CONTENTS, LOCK) ;
            GOTOXY (30, 3) ;
            WRITE ('Strike <return>.') ;
            READ (KEYIN) ;
           end ;   (* if CONTENTS^.CHAP <>0 *)
      end ;   (* end '0' *)
'S' : begin
       CLOSE (CONTENTS, LOCK) ;
```

```
             CLOSE (LIBRARY, LOCK) ;
             EXIT (CRTBOOKS) ;
            end ;     (* 'S' *)
    end ;
   until FALSE ;
   CLOSE (CONTENTS, LOCK) ;
   CLOSE (LIBRARY, LOCK) ;
  end ;  (* CRTBOOKS *)



procedure CRTDATA ;
  var
     WD, COL, CLMN : integer ;
     DWORD   : STRKEY ;
     DISPLAY : boolean ;
     DISPLYF : boolean ;
     FSTR    : FNAMES ;

  procedure LOOKUP  (var DATAWORD : STRKEY) ;
   var
      NODE, I : integer ;
   begin   (* LOOKUP *)
    NODE := AVAILIST^.ROOT ;
    SEEK (DICTION, NODE) ;
    GET (DICTION) ;
    DISPLYF := TRUE ;
    I := 1 ;
    with DICTION^ do
     if (not FLAG) and (POINTER[ I ] = NULL)
```

```
      then
       begin
             DATAWORD := COPY (BLANK, 1, MAXKEY) ;
             SETPOSITION ;
             WRITE ('No data words in DATA DICTIONARY<ret>.' ) ;
             READ (KEYIN) ;
             SETPOSITION ;
             DISPLYF := FALSE ;
             CLOSE (DICTION, LOCK) ;
             CLOSE (AVAILIST, LOCK) ;
             EXIT ( CRTDATA ) ;
       end
      else
       begin
        while not FLAG do
         begin
          NODE := POINTER [ I ] ;
          SEEK (DICTION, NODE) ;
          GET (DICTION) ;
         end ;
        DATAWORD := SUBKEY[ I ].WKEY ;
        CHAPHEAD := SUBKEY[ I ].CHAP ;
        BOOK     := SUBKEY[ I ].TITLE ;
        CLMN     := SUBKEY[ I ].COLMN ;
       end ;
 end ;      (* LOOKUP *)

procedure NEXTWORD (var WORD : integer ; var DATAWORD : STRKEY) ;
 begin    (* NEXTWORD *)
  DISPLAY := TRUE ;
 DISPLYF := TRUE ;
 WORD := WORD + 1 ;
 with DICTION^ do
  begin
   if SUBKEY[ WORD ].WKEY = COPY (BLANK, 1, MAXKEY)
      then if LINK = NULL
            then
             begin
              DISPLAY := FALSE ;
              DISPLYF := FALSE ;
              PAGE (OUTPUT) ;
              SETPOSITION ;
              WRITE ('No more words. <ret>.') ;
              READ (KEYIN) ;
              EXIT (NEXTWORD) ;
             end     (* then *)
            else
             begin
              SEEK (DICTION, LINK) ;
              GET (DICTION) ;
              WORD := 1 ;
             end ;    (* else *)
    DATAWORD := SUBKEY[ WORD ].WKEY ;
    CHAPHEAD := SUBKEY[ WORD ].CHAP ;
    BOOK     := SUBKEY[ WORD ].TITLE ;
    CLMN     := SUBKEY[ WORD ].COLMN ;
   end ;   (* with *)
end ;      (* NEXTWORD *)
```

```
procedure DATWORD (var WORD : integer ; var DATAWORD : STRKEY) ;
 var
    I, NODE : integer ;
    FOUND   : boolean ;
 begin     (* DATWORD *)
  PAGE (OUTPUT) ;
  GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ; GOTOXY (LEFT, UPPER) ;
  WRITE ('Enter data word : ') ;
  READLN (DATAWORD) ;
  NODE := AVAILIST^.ROOT ;
  FOUND := FALSE ;
  repeat
   I := 1 ;
   SEEK (DICTION, NODE) ;
   GET (DICTION) ;
   with DICTION^ do
    begin
     while (SUBKEY[ I ] .WKEY <> COPY (BLANK, 1, MAXKEY) )
           and (SUBKEY[ I ].WKEY < DATAWORD) do I := SUCC ( I ) ;
     if (SUBKEY[ I ].WKEY = DATAWORD) and (FLAG)
        then
         begin
          DISPLAY := TRUE ;
          DISPLYF := TRUE ;
          WORD := I ;
          FOUND := TRUE ;
         end ;
     NODE := POINTER[ I ] ;
    end ;
  until (NODE = NULL) or (DICTION^.FLAG) ;
  if not FOUND
     then
      begin
       DISPLAY := FALSE ;
       DISPLYF := FALSE ;
       SETPOSITION ;
       WRITE (DATAWORD, ' not found. <ret>.') ;
       READ (KEYIN) ;
       EXIT (DATWORD) ;
      end ;
  CHAPHEAD := DICTION^.SUBKEY[ WORD ].CHAP ;
  BOOK     := DICTION^.SUBKEY[ WORD ].TITLE ;
  CLMN     := DICTION^.SUBKEY[ WORD ].COLMN ;
 end ;   (* DATWORD *)

function WRONGPW ( var STRNG : FNAMES) : boolean ;
 var
    I, IT : integer ;
 begin
  WRONGPW := FALSE ;
  RESET (LIBRARY, CONCAT (LIBNAME, '.LIB') ) ;
  GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ;
  GOTOXY (LEFT, UPPER) ;
  WRITE ('Enter book password : ') ;
  READLN (KEYBOARD, LINEIN) ;
  PW := LINEIN ;
  with LIBRARY^ do
   begin
    for I := 1 to ITEMS do if TITLES[ I ] = BOOK then IT := I ;
    if PASSWORD[ IT ] <> PW
```

```
        then
         begin
          WRONGPW := TRUE ;
          STRNG   := 'book' ;
          CLOSE (LIBRARY, LOCK) ;
          EXIT (WRONGPW) ;
         end ;
   end ;          (* with *)
  CLOSE (LIBRARY, LOCK) ;
  RESET (CONTENTS, CONCAT( PREFIX, BOOK, '.BOOK') ) ;
  GOTOXY (LEFT, UPPER) ; WRITE (BLANK) ;
  GOTOXY (LEFT, UPPER) ;
  WRITE ('Enter chapter password : ') ;
  READLN (KEYBOARD, LINEIN) ;
  PW := LINEIN ;
  with CONTENTS^ do
   begin
    for I := 1 to NCHAP do if HEADING[ I ] = CHAPHEAD then IT := I ;
    if PASWORD[ IT ] <> PW
       then
        begin
         WRONGPW := TRUE ;
         STRNG   := 'chapter' ;
         CLOSE (CONTENTS, LOCK) ;
         EXIT (WRONGPW) ;
        end ;
  end ;         (* with *)
  CLOSE (CONTENTS, LOCK) ;
end ;       (* WRONGPW *)
```

```
begin              (* CRTDATA *)
 PAGE (OUTPUT) ;
 RESET (DICTION, CONCAT (LIBNAME, '.DICT') ) ;
 RESET (AVAILIST, CONCAT (LIBNAME, '.AVAIL') ) ;
 LOOKUP (DWORD) ;     (* find smallest data word *)
 WD := 1 ;
 DISPLAY := FALSE ;
 SETPOSITION ;
 WRITE (DWORD, '=1st data word in DATA DICT<ret>') ;
 READ (KEYIN) ;
 WRITE ('Bk:', COPY (BOOK, 1, LENGTH(BOOK) ),
          ',ch:', COPY (CHAPHEAD, 1, LENGTH(CHAPHEAD) ),
          ',cl:', CLMN:2, '.') ;
 repeat
  case BANNER ('F(orm, N(ext word, any D(ata word or S(top:') of
   'F' : if DISPLYF
          then
           begin
            if WRONGPW (FSTR) then begin
                             SETPOSITION ;
                             WRITE ('Wrong ', COPY (FSTR, 1, LENGTH(FSTR) ),
                                    ' password. <ret>.') ;
                             READ (KEYIN) ;
                             SETPOSITION ;
                             CLOSE (DICTION, LOCK) ;
                             CLOSE (AVAILIST, LOCK) ;
                             EXIT (CRTDATA) ;
                            end ;
            PAGE (OUTPUT) ; SETPOSITION ;
            WRITE ( 'Bk:', BOOK, ',ch:', CHAPHEAD,
                    ',cl:', CLMN:2, '. <ret>.') ;
           READ (KEYIN) ;
           RESET (FORM, CONCAT ( PREFIX, CHAPHEAD, '.FORM') ) ;
           DISPLAY := FALSE ;
           if FORM^.NCOL > 1 then
            repeat
             SETPOSITION ;
             WRITE ('Enter col. no. of page form ',
                    '1..', FORM^.NCOL:2, ' : ') ;
             READLN (COL) ;
            until (COL > 0) and (COL <= FORM^.NCOL) ;
           SHOWFORM (CHAPHEAD, COL) ;
           CLOSE (FORM, LOCK) ;
          end
         else
          begin
           SETPOSITION ;
           WRITE ('No form to be displayed. <ret>.') ;
           READ  (KEYIN) ;
           SETPOSITION ;
          end ;
  'N' : NEXTWORD (WD, DWORD) ;
  'D' : DATWORD  (WD, DWORD) ;
  'S' : begin
         CLOSE (DICTION, LOCK) ;
         CLOSE (AVAILIST, LOCK) ;
         EXIT (CRTDATA) ;
        end ; (* 'S' *)
 end ;        (* case *)
if DISPLAY
   then
```

```
       begin
        PAGE (OUTPUT) ; SETPOSITION ;
        WRITE ( 'Bk:', BOOK, ',ch:', CHAPHEAD,',cl:',
               CLMN:2, '. <ret>.') ;
        READ (KEYIN) ;
       end ;
  until FALSE ;
  CLOSE (DICTION, LOCK) ;
  CLOSE (AVAILIST, LOCK) ;
 end ;  (* CRTDATA *)

begin   (* BODY GLANCE *)
repeat
   PAGE (OUTPUT) ;
   case BANNER ('Display : B(ook, D(ata, or S(top : ') of
    'B' : CRTBOOKS ;
    'D' : CRTDATA ;
    'S' : begin
           CLOSE (LIBRARY, LOCK) ;
           EXIT ( GLANCE ) ;
          end ;  (* 'S'  *)
   end ;  (* case *)
  until FALSE ;
end ;   (*  GLANCE *)


 procedure GENASYS ;
  var
   SIZE, ITEM, TSIZE : integer ;
```

```
procedure BOX ;

 Begin
  PAGE( OUTPUT ) ;
  GOTOXY( LEFT, UPPER ) ; WRITE( 'Screen size :' ) ;
  for TSIZE := 1 to YLIMIT do
   begin
    GOTOXY( XORIGIN, UPPER+TSIZE+1 ) ; WRITE( '!' );
    GOTOXY( XLIMIT+1, UPPER+TSIZE+1 ) ; WRITE( '!' );
   end ;
  for TSIZE := XORIGIN to XLIMIT+1 do
   Begin GOTOXY( TSIZE, YLIMIT+1 ) ; WRITE( '.' ) ; end ;
  GOTOXY( LEFT, UPPER+1 ) ; WRITELN( 'Strike <rtn>.' ) ;
  READ( KEYIN ) ;
  GOTOXY( LEFT, UPPER+4 ) ;
  WRITELN( 'Max. # books    = ', NBOOKS );
  WRITELN( 'Max. # chapters = ', NCHAPS ) ;
  WRITELN( 'Max. # columns  = ', NCOLS  ) ;
  WRITELN( 'Max. words/page = ', NWORDS );
  WRITELN( 'Max. word length= ', MAXWORD) ;
  WRITELN( 'Max. book/chap name length = ', MAXCHAP ) ;
  WRITELN( 'Width of screen = ', XLIMIT ) ;
  WRITELN( 'Length of screen= ', YLIMIT ) ;
  WRITELN( 'Available memory space = ', MEMAVAIL ) ;
  WRITELN( 'These can be changed by re-compiling, only' );
 end; (*    BOX     *)


procedure HEADING ;
 begin
```

```
  PAGE( OUTPUT ) ;
  GOTOXY ( LEFT+5, UPPER ); WRITE( 'C R E A T I N G   M I C R O B O O K' );
  GOTOXY( LEFT, UPPER+1 ) ; WRITE ( 'Enter max. # words in dictionary = ' ) ;
  READLN( SIZE ) ;
  SIZE := SIZE div  NBTREE + 1;
  NPX := 1 ; repeat NPX := 2 * NPX ; SIZE := SIZE - 1 until SIZE < 1 ;
  SIZE := NPX ;
  REWRITE( LIBRARY, CONCAT( LIBNAME,'.LIB' ) ) ;
  with LIBRARY^ do
   begin
    ITEMS     := 0 ;
    for ITEM := 1 to NBOOKS do
      begin
       TITLES[ ITEM ] := SHORT ;
       PASSWORD[ ITEM ] := SHORT ;
     end ;
   end ; (* with *)
 SEEK( LIBRARY, 0 ) ;
 PUT( LIBRARY ) ;
 CLOSE( LIBRARY, LOCK ) ;
end ;    (*    HEADING    *)

 begin
  HEADING ;
  REWRITE( DICTION, CONCAT (LIBNAME, '.DICT') ) ;
  DICTION^.FLAG := FALSE ;
  DICTION^.LINK := NULL ;
  for ITEM := 1 to NBTREE do
     with DICTION^ do
      begin
```

```
       POINTER[ ITEM ] := NULL ;
       with SUBKEY[ ITEM ] do
        begin
         WKEY := COPY( BLANK, 1, MAXKEY ) ;
         TITLE := SHORT ;
         CHAP := SHORT ;
         COLMN:= YORIGIN ;
         XPOS := XORIGIN ; YPOS := YORIGIN ; LONG := XORIGIN ;
         PLACE:= 0 ;
        end ;
      end ;
    for TSIZE := 0 to SIZE do
      begin
       SEEK( DICTION, TSIZE ) ;
       PUT( DICTION ) ;
      end ;
    CLOSE( DICTION, LOCK ) ;
    REWRITE( AVAILIST, CONCAT (LIBNAME, '.AVAIL') ) ;
    AVAILIST^.TN := 0 ;
    AVAILIST^.TREESIZE := SIZE ;
    AVAILIST^.ROOT := 0 ;
    SEEK( AVAILIST, 0 ) ;
    PUT( AVAILIST ) ;
    CLOSE( AVAILIST, LOCK ) ;
    BOX ;
    GOTOXY( LEFT, UPPER+1  ) ; WRITE('Strike <rtn>.'); READ( KEYIN ) ;
end ;  (* GENASYS *)
```

```
procedure  COPYDB  ;
const
   MIDDLE  = 25 ;
   FOOTING = 16 ;
   FIRST   = 1;
   LASTROOT = 1 ;
type

   QUEUE       = record
                  IT      : integer ;
                  TITLE   : array [1..NBOOKS ] of FNAMES ;
                  PSW     : array [1..NBOOKS ] of FNAMES ;
                 end ;
   var
      BLANKWORDS, KEY   : WORDS ;
      BLANKIDX          : PAGEDATA ;
      NEWLIB            : file of BOOKS ;
      NEWCONT           : file of CHAPTERS ;
      NEWFORM           : file of PAGES  ;
      NEWAVAIL          : file of EMPTIES ;
      NEWDICT           : file of NODES ;
      NEWPIDX           : file of PAGEDATA ;
      NEWAIDX           : file of PUSED ;
      NEWPPG            : file of DATUM ;
      NDRIVE , FILER    : string ;
      TSIZE             : integer;
      WORSIZE , ITEM    : integer ;
      VWORDS            : integer ;
      WORKQUE           : QUEUE ;
      DUMMYJOB          : boolean ;
      NOMORE              : boolean ;
      BI,  BCNT           : 1..NBOOKS ;
      CI,  CCNT           : 1..NCHAPS ;
   procedure  FOOTAREA ;
      begin
         GOTOXY (LEFT,FOOTING - 1 ) ;
         WRITELN ( CONCAT ( '----------------------------------------' ,
                           '----------------------------------------'  ) ) ;
     end ;

     (************************* A C C E P T ******************)

procedure  ACCEPT (var   WORKQUE : QUEUE ;  MAXCNT : integer  ) ;

const
  RSIDE = 32 ;

var
   PASS   : FNAMES  ;
   IDX : integer ;                (* screen line pointer *)
   WIDX  : integer ;              (* workque array index *)
   SIDX :  integer ;              (*shift to left pointer*)
   STOREQUE  : QUEUE ;
begin
   STOREQUE := WORKQUE ;
   repeat
      WORKQUE := STOREQUE ;
      WIDX := 1 ;
      GOTOXY (27, FIRST) ;
      WRITELN ( 'S(ave  D(elete ?');
for  IDX := 1      to  MAXCNT    do
   begin
      GOTOXY (RSIDE, FIRST + IDX ) ;
      READ (KEYIN) ;
      if  KEYIN = 'S'   then
         begin
            GOTOXY (RSIDE, FIRST + IDX ) ;
            WRITE ( 'PASSWORD :') ;
            READLN (KEYBOARD, PASS ) ;
            if  PASS <> STOREQUE.PSW [ IDX ]  then
               begin
                  GOTOXY (RSIDE, FIRST + IDX );
                  WRITE ( 'try again :' ) ;
                  READLN (KEYBOARD, PASS ) ;
               end ;
            GOTOXY ( RSIDE, FIRST + IDX ) ;
            if  PASS <> STOREQUE.PSW [ IDX ]   then
               begin
                  WRITE ('Delete         ');
                  KEYIN := 'D' ;
               end
            else
               WRITE ('Save           ') ;
         end ;
      if  KEYIN  <>   'D'   then
          WIDX := WIDX + 1
       else
          begin
             with WORKQUE    do
                begin
```

```
                         IT := IT - 1 ;
                         if  IT > 0    then
                            begin
                               for  SIDX := WIDX      to  IT        do
                                  begin
                                     TITLE [ SIDX ] := TITLE [ SIDX + 1 ] ;
                                     PSW [ SIDX ] := PSW [ SIDX + 1 ] ;
                                   end;
                            end;
                       end;
                  end;
                end ;
               GOTOXY (LEFT,UPPER );
               WRITE ( 'Are you sure ? ');
               READ (KEYIN);
   until  KEYIN = 'Y' ;
end ;                                  (* ACCEPT *)

procedure SHOW ( FROM, TILL, BACK : integer ) ;

var
   I :  integer ;

begin
   for  I := FROM  to   TILL    do
      begin
         GOTOXY ( LEFT, I ) ;
         WRITELN ( BLANK );
      end ;
   GOTOXY (LEFT , BACK ) ;
end ;
```

```
       (****************** U P D I C T *************************)
procedure UPDICT ;
 var
    WORKCOUNT : integer ;
    WORKAREA  : NODES ;
    ITEM      : 1..NBTREE ;
    SPOINT, GOON, REST  : integer ;
 begin
    NOMORE := FALSE ;
    WORKAREA.FLAG := FALSE ;
    SEEK (NEWAVAIL, 0);
    SEEK (NEWDICT, 0) ;
    GET (NEWDICT) ;
    if  NEWDICT^.LINK  <> NULL  then
       begin
          repeat
            WORKCOUNT := 0 ;
            SEEK (NEWDICT,NEWAVAIL^.ROOT) ;
            GET (NEWDICT ) ;
            NEWAVAIL^.ROOT := NEWAVAIL^.TN + 1 ;
            ITEM := 1 ;
            while  NEWDICT^.LINK <> NULL do
               begin
                  SPOINT := NEWDICT^.LINK - 1 ;
                  with  WORKAREA  do
                     begin
                        SUBKEY [ ITEM ] := NEWDICT^.SUBKEY [ NBTREE - 1 ] ;
                        POINTER [ ITEM ] := SPOINT ;
                     end ;
                  ITEM := ITEM + 1 ;
```

```
if  ITEM = NBTREE    then
   begin
      ITEM := 1 ;
      NEWAVAIL^.TN := NEWAVAIL^.TN + 1 ;
      WORKAREA.LINK := NEWAVAIL^.TN + 1 ;
      NEWDICT^ := WORKAREA ;
      SEEK (NEWDICT, NEWAVAIL^.TN) ;
      PUT (NEWDICT );
      WORKCOUNT := WORKCOUNT + 1 ;
      GOON := SPOINT + 1 ;
      SEEK (NEWDICT,GOON);
      GET (NEWDICT ) ;
   end                         (** IF ITEM = NBTREE **)
else
   begin

      SEEK (NEWDICT,NEWDICT^.LINK) ;
      GET (NEWDICT);

   end ;
end ;
with   WORKAREA     do
   begin
      POINTER [ ITEM ] := SPOINT + 1 ;
      SUBKEY [ ITEM ] := BLANKWORDS ;
      for REST := ITEM + 1    to  NBTREE do
         begin
            POINTER [ REST ] := NULL ;
            SUBKEY [ REST ] := BLANKWORDS ;
         end ;
```

```
                        ITEM := 1 ;
                        LINK := NULL ;
                     end (* with *) ;
                     NEWAVAIL^.TN := NEWAVAIL^.TN + 1 ;
                     NEWDICT^ := WORKAREA ;
                     SEEK (NEWDICT,NEWAVAIL^.TN);
                     PUT (NEWDICT) ;
                     WORKCOUNT := WORKCOUNT + 1 ;
                     if  WORKCOUNT = LASTROOT   then
                        NOMORE  :=  TRUE
                     else
                        NEWAVAIL^.ROOT :=  NEWAVAIL^.TN + 1 ;
       until  NOMORE ;
       NEWAVAIL^.ROOT := NEWAVAIL^.TN ;
       SEEK (NEWAVAIL, 0) ;
       PUT ( NEWAVAIL ) ;
  end ;                               (** end if **)
  CLOSE (NEWAVAIL,LOCK );
  CLOSE (NEWDICT,LOCK ) ;
end ;                                 (** updict   **)


procedure  GETPAGEFORM ;
var
   COL, MAXCOL : NULL..NCOLS ;
   DPENO        : integer ;             (* page # in a chapter   *)
   DPONE        : integer ;             (* page # in a column 1  *)
   DPTWO        : integer ;             (* page # in a column 2  *)
   SAVEPN : integer ;
   CHAPTER : FNAMES ;
   PACKTREE : boolean ;
```

```
(**************************** D I C T I O N A R Y *****************)

procedure  DICTIONARY ;

 var

    M :  1..NWORDS ;
    TREEN : integer ;
    K :  integer ;
    I : 1..NBTREE ;
    KEY : string [ MAXKEY ];
    DATAWORD : WORDS ;

procedure  SQUZDICT  ;
 begin
  repeat
   NOMORE := TRUE ;

    with  NEWDICT^  do
      begin
        IF  (SUBKEY [ NBTREE - 1 ].WKEY = COPY ( BLANK , 1, MAXKEY ) )
                  or
            (SUBKEY [ NBTREE - 1 ].WKEY > KEY )       then
            begin
               I := FIRST ;
               while ( SUBKEY [ I ].WKEY <> COPY (BLANK ,1, MAXKEY ) )
                       and
                     ( SUBKEY [ I ].WKEY  < KEY )       do
                   I  := I + 1 ;
                for   K := NBTREE  downto  I + 1    do
```

```
         SUBKEY [ K ] := SUBKEY [ K-1 ] ;
      SUBKEY [ I ] := DATAWORD ;
      FLAG := TRUE ;
      if  SUBKEY [ NBTREE ].WKEY <> COPY (BLANK, 1, MAXKEY )   then
         begin
            KEY := SUBKEY[ NBTREE ].WKEY ;
            DATAWORD := SUBKEY [ NBTREE ] ;
            SUBKEY [ NBTREE ] := BLANKWORDS ;
            if  LINK = NULL   then
               begin
                  TREEN := TREEN + 1 ;
                  LINK := TREEN ;
               end ;
            PUT  (NEWDICT ) ;
            SEEK (NEWDICT, LINK ) ;
            GET (NEWDICT ) ;
            SEEK (NEWDICT,TREEN);
            NOMORE := FALSE ;
         end ;
   end  (* if *)
else
    begin
       if  LINK = NULL    then
          begin
             TREEN := TREEN + 1 ;
             LINK := TREEN ;
          end ;
       PUT (NEWDICT ) ;
       SEEK (NEWDICT, LINK ) ;
       GET (NEWDICT ) ;
```

```
                 SEEK (NEWDICT,TREEN) ;

                 NOMORE := FALSE ;
             end;                     (* else     *)
         end;                         (* with     *)
       until NOMORE ;
     end;                             (* SQUZDICT *)

 begin                (** DICTIONARY **)
    SEEK ( NEWAVAIL,0 );
    GET ( NEWAVAIL ) ;
    TREEN := NEWAVAIL^.TN ;
    for  M := 1   to  NEWFORM^.NENTRIES   do
       begin
          KEY := NEWFORM^.ENTRY [ M ].WKEY ;
          DATAWORD := NEWFORM^.ENTRY [ M] ;
          SEEK (NEWDICT, NEWAVAIL^.ROOT ) ;
          GET (NEWDICT ) ;
          SEEK (NEWDICT, NEWAVAIL^.ROOT );
          SQUZDICT  ;
          PUT (NEWDICT ) ;
       end ;
    NEWAVAIL^.TN := TREEN ;
    SEEK (NEWAVAIL, 0) ;
    PUT (NEWAVAIL ) ;
 end ;                                  (* DICTIONARY *)

   (****************** INDEXFILE ************************)
procedure  INDEXFILE ;
 var
```

```
    BTREE :  FNAMES ;
    DATAWORD :  PLOCATION ;
    BLANKPLAC  : PLOCATION ;
    KEY :  string [ MAXWORD ];
    I :  1..NPP ;
    K :  integer ;
    TREEP : integer ;


procedure  SQUEEZEIDX ;

   begin
    repeat
     NOMORE := TRUE ;

      with  NEWPIDX^    do
         begin
            if  (PPSUBKEY [ NPP - 1 ].PKEY =   COPY (BLANK , 1 , MAXWORD) )
                    or
                (PPSUBKEY [ NPP - 1 ].PKEY > KEY )      then
               begin
                  I := FIRST ;
                  while  (PPSUBKEY [ I].PKEY <> COPY (BLANK, 1, MAXWORD))
                            and
                         (PPSUBKEY [ I ].PKEY < KEY )   do
                     I := I + 1 ;
                  for   K:= NPP      downto   I + 1    do
                     PPSUBKEY [ K ] := PPSUBKEY [ K - 1 ] ;
                  PPSUBKEY [ I ] := DATAWORD ;
                  PFLAG := TRUE ;
   if (  PPSUBKEY [NPP ].PKEY <> COPY (BLANK, 1, MAXWORD))
      then
       begin
          KEY := PPSUBKEY [ NPP ].PKEY ;
          DATAWORD := PPSUBKEY [ NPP ] ;
          PPSUBKEY [ NPP ] := BLANKPLAC ;
          if  PLINK = NULL      then
             begin
                TREEP := TREEP +1 ;
                PLINK := TREEP ;
             end ;
          PUT (NEWPIDX) ;
          SEEK (NEWPIDX,PLINK);
          GET ( NEWPIDX ) ;
          SEEK (NEWPIDX,TREEP) ;
          NOMORE := FALSE ;
        end;                                (**** end if **)
  end                                           (**** end if **)
else
      begin
         IF PLINK = NULL  then
            begin
               TREEP := TREEP + 1 ;
               PLINK := TREEP ;
            end ;
         PUT ( NEWPIDX ) ;
         SEEK (NEWPIDX,PLINK) ;
         GET ( NEWPIDX ) ;
         SEEK (NEWPIDX,TREEP) ;
         NOMORE := FALSE ;
```

```
                     end;                                     (*** end else**)
                 end;                                       (*****end with **)
                until NOMORE ;
              end;                                          (**squeezeidx **)

procedure UPPIDX ;
 var
    WORKCOUNT : integer ;
    WORKAREA  : PAGEDATA ;
    ITEM      : 1..NPP ;
    NOMORE    : boolean ;
    SPOINT, GOON, REST  : integer ;

 begin
    NOMORE := FALSE ;
    WORKAREA.PFLAG := FALSE ;
    SEEK (NEWPIDX, 0) ;
    GET (NEWPIDX) ;
    if  NEWPIDX^.PLINK  <> NULL  then
       begin
          repeat
             WORKCOUNT := 0 ;
             SEEK (NEWPIDX,NEWAIDX^.PROOT) ;
             GET (NEWPIDX ) ;
             NEWAIDX^.PROOT := NEWAIDX^.PN + 1 ;
             ITEM := 1 ;
             while  NEWPIDX^.PLINK <> NULL do
                begin
                   SPOINT := NEWPIDX^.PLINK - 1 ;
                   with  WORKAREA  do
```

```
   begin
      PPSUBKEY [ ITEM ] := NEWPIDX^.PPSUBKEY [ NPP - 1 ] ;
      PPOINTER [ ITEM ] := SPOINT ;
   end ;
ITEM := ITEM + 1 ;
if  ITEM = NPP       then
   begin
      ITEM := 1 ;
      NEWAIDX^.PN := NEWAIDX^.PN  + 1 ;
      WORKAREA.PLINK := NEWAIDX^.PN  + 1 ;
      NEWPIDX^ := WORKAREA ;
      SEEK (NEWPIDX, NEWAIDX^.PN ) ;
      PUT (NEWPIDX );
      WORKCOUNT := WORKCOUNT + 1 ;
      GOON := SPOINT + 1 ;
      SEEK (NEWPIDX,GOON);
      GET (NEWPIDX ) ;
   end                         (** IF ITEM = NPP  **)
else
   begin

      SEEK (NEWPIDX,NEWPIDX^.PLINK) ;
      GET (NEWPIDX);

   end ;
end  (* while *);
with   WORKAREA     do
   begin
      PPOINTER [ ITEM ] := SPOINT + 1 ;
      PPSUBKEY [ ITEM ] := BLANKPLAC  ;
```

```
                         for REST := ITEM + 1   to  NPP do
                            begin
                               PPOINTER [ REST ] := NULL ;
                               PPSUBKEY [ REST ] := BLANKPLAC  ;
                            end ;
                         ITEM := 1 ;
                         PLINK := NULL ;
                      end ; (* with *)
                      NEWAIDX^.PN  := NEWAIDX^.PN  + 1 ;
                      NEWPIDX^ := WORKAREA ;
                      SEEK (NEWPIDX,NEWAIDX^.PN );
                      PUT (NEWPIDX) ;
                      WORKCOUNT := WORKCOUNT + 1 ;
                      if  WORKCOUNT = LASTROOT   then
                         NOMORE  :=  TRUE
                      else
                         NEWAIDX^.PROOT :=  NEWAIDX^.PN  + 1 ;
                   until  NOMORE ;
                NEWAIDX^.PROOT := NEWAIDX^.PN  ;
                SEEK (NEWAIDX , 0 );
                PUT ( NEWAIDX  ) ;
          end ;                                  (** end if  **)
          CLOSE (NEWAIDX ,LOCK );
          CLOSE (NEWPIDX,LOCK ) ;
       end ;                                      (** uppidx   **)

procedure   MAKEIDX ;
  begin
     with  NEWPIDX^    do
       begin
```

```
          for   I := 1    to   NPP       do
             begin
                with  PPSUBKEY [ I ]     do
                   begin
                      PKEY := COPY (BLANK, 1 , MAXWORD ) ;
                      PTITLE := COPY (BLANK , 1, MAXKEY ) ;
                      PHEAD := COPY ( BLANK, 1, MAXKEY ) ;
                      PP    := 0 ;
                      PCOLUMN := 1 ;
                   end ;           (* end with-ppsubkey *)
                 PPOINTER [ I ] := NULL ;
              end ;                (*end for *)
        PLINK := NULL ;
        PFLAG := FALSE ;
    end;                           (* end with-newpidx *)
 if  PACKTREE      then
     VWORDS := NODECOUNT ( NEWFORM^.MAXPAGE, NPP - 1 )
 else
    VWORDS := NODECOUNT (NEWFORM^.MAXPAGE, TRUNC ( ( NPP - 1) / 2 ) + 1 ) ;
 SHOW (UPPER,8,UPPER ) ;
 WRITELN ( VWORDS, ' records are available for ', NDRIVE,BTREE  ,'.INDEX');
 for   I := 1   TO  VWORDS  do
    begin
       SEEK ( NEWPIDX , I - 1 ) ;
       PUT (NEWPIDX ) ;
    end ;
 BLANKPLAC := NEWPIDX^. PPSUBKEY [ NPP ] ;
 CLOSE (NEWPIDX, LOCK ) ;
end ;                                           (* MAKEIDX *)
```

```
begin                                  (** INDEXFILE **)
   with  NEWFORM^  do
    begin
      for  ITEM := 1   to  NENTRIES   do
        begin
          if  POS ('*', ENTRY [ ITEM ].WKEY ) = 1      then
            begin
               TREEP := 0 ;
               BTREE := COPY (ENTRY [ITEM].WKEY, 2,
                          ( LENGTH (ENTRY [ITEM].WKEY )  - 1 ) );
               RESET (NEWPPG,CONCAT (NDRIVE, CHAPTER, '.PAGES' ) ) ;
               SHOW ( FOOTING + 3, FOOTING + 3, FOOTING + 3) ;
               WRITELN ('** BACKUP : ' , BTREE, '.INDEX');
               REWRITE (NEWPIDX,CONCAT( NDRIVE,BTREE, '.INDEX') ) ;
               MAKEIDX ;
               REWRITE (NEWAIDX,CONCAT (NDRIVE,BTREE, '.AVAIL ' ) ) ;
               SHOW ( FOOTING + 3, FOOTING + 3, FOOTING + 3) ;
               WRITELN ('** BACKUP : ', BTREE, '.AVAIL') ;
               NEWAIDX^.PN := 0 ;
               NEWAIDX^.PROOT := 0 ;
               SEEK (NEWAIDX, 0);
               PUT (NEWAIDX) ;
               CLOSE (NEWAIDX,LOCK ) ;
               RESET (NEWPIDX, CONCAT (NDRIVE,BTREE,'.INDEX' ) ) ;
               RESET (NEWAIDX, CONCAT (NDRIVE,BTREE,'.AVAIL' ) ) ;
               while  ( not EOF (NEWPPG) ) and  (NEWPPG^.DPAGENO <> NULL ) do
                  begin
                     if  NEWPPG^.DCOL = COL  then
                        begin
                           with  DATAWORD   do
```

```
                              begin
                                 PKEY := NEWPPG^.VALUES [ ITEM ] ;
                                 PTITLE := BOOK ;
                                 PHEAD := CHAPTER ;
                                 PP := NEWPPG^.DPAGENO;
                                 PCOLUMN := COL ;
                                 KEY := PKEY ;
                              end;
                           SEEK (NEWPIDX, NEWAIDX^.PROOT);
                           GET (NEWPIDX) ;
                           SEEK (NEWPIDX, NEWAIDX^.PROOT);
                           SQUEEZEIDX ;
                           PUT (NEWPIDX ) ;
                        end ; (* if *)
                     GET (NEWPPG);
                  end ; (* while *)
                  NEWAIDX^.PN := TREEP ;
                  SEEK (NEWAIDX, 0) ;
                  PUT (NEWAIDX) ;
                  UPPIDX ;
                  CLOSE (NEWPPG,LOCK);
              end ;       (* if *)
          end ;           (* for *)
      end ;               (* with *)
   end;                   (* INDEXFILE *)

(***********************  G E T P A D E F O R M  ********************)

procedure  SETPAGE ;
 var
```

```
   I :  integer ;
begin
   repeat
      PACKTREE := TRUE  ;
      SHOW (UPPER,5 , UPPER) ;
      WRITELN (DPONE + DPTWO,' records are copied to ', CHAPTER, '.PAGES') ;
      WRITELN ( ' ***  column 1 : ' , DPONE, ' records');
      WRITELN ( ' ***  column 2 : ' , DPTWO, ' records');
      WRITE   ( 'enter number for more pages or 0 to stop ') ;
      READLN ( VWORDS ) ;
      SHOW (UPPER,5, UPPER) ;
      WRITE ( 'Are you sure ? ') ;
      READLN (KEYIN ) ;
   until  (KEYIN  =  'Y');
   SHOW (UPPER, UPPER,UPPER ) ;
   with  NEWPPG^    do
      begin
         DSTAT := TRUE ;
         DPAGENO := NULL ;
         DCOL := YORIGIN;
         for   I := 1    to  NWORDS  do
               VALUES [ I ] := COPY ( BLANK , 1, MAXWORD ) ;
      end ;(* with *)
   if  VWORDS + DPENO = NULL   then
         PUT (NEWPPG) ;             (* GIVE A DUMMY RECORD FOR DUMMY PAGE FILE *)
   if  VWORDS   >  0   then
      begin
         PACKTREE := FALSE ;
         if  ( DPONE = 0 ) or ( DPTWO = 0 )  then
                 SAVEPN := VWORDS
```

```
           else
                SAVEPN := VWORDS * 2 ;
           for  I := 1      to   SAVEPN    do
                   PUT (NEWPPG) ;
       end ; (* if *)
    SHOW (FOOTING + 1, FOOTING + 1, FOOTING + 1) ;
    WRITELN ('** BACKUP : ' , CHAPTER, '.FORM' ) ;
    CLOSE ( NEWPPG,LOCK ) ;
    REWRITE (NEWFORM, CONCAT (NDRIVE, CHAPTER, '.FORM') ) ;
    RESET ( FORM, CONCAT (PREFIX, CHAPTER, '.FORM' ) );
    MAXCOL := FORM^.NCOL ;
    if  DPENO = NULL    then   DPENO :=  0 ;
    for  COL :=  1         to   MAXCOL    do
       begin
          SEEK (FORM, COL - 1 ) ;
          GET (FORM ) ;
          NEWFORM^ := FORM^ ;
          NEWFORM^.MAXPAGE :=  DPENO + VWORDS ;
          SEEK (NEWFORM, COL - 1) ;
          PUT ( NEWFORM ) ;
       end ;
    CLOSE (NEWFORM , LOCK ) ;
    RESET (NEWFORM, CONCAT (NDRIVE,CHAPTER, '.FORM' ) ) ;
end ; (* SETPAGE *)

begin             (* GETPAGEFORM *)
   CHAPTER := NEWCONT^.HEADING [ CI ] ;
   REWRITE (NEWPPG, CONCAT (NDRIVE,CHAPTER, '.PAGES') ) ;
   SEEK (NEWPPG, 0) ;
   RESET   (PPG, CONCAT (PREFIX,CHAPTER, '.PAGES') ) ;
```

```
FOOTAREA ;
SHOW ( FOOTING + 1, FOOTING + 3, FOOTING + 1) ;
WRITELN ( '** BACKUP : ' , CHAPTER, '.PAGES' ) ;
WRITELN (BLANK);
WRITELN (BLANK) ;
SAVEPN := NULL ;
DPENO  := NULL ;
DPONE  := 0    ;
DPTWO  := 0    ;
while ( not EOF (PPG) )  and  ( PPG^.DPAGENO <> NULL )    do
   begin
      if PPG^.DSTAT  then
         begin
           if  PPG^.DCOL = 1   then
                  DPONE := DPONE + 1 ;
           if  PPG^.DCOL = 2   then
                 DPTWO  := DPTWO + 1;
           if  PPG^.DPAGENO <> SAVEPN      then
              begin
                 DPENO := DPENO + 1 ;
                 SAVEPN := PPG^.DPAGENO;
              end ;
           NEWPPG^ := PPG^ ;
           NEWPPG^.DPAGENO := DPENO ;
           PUT (NEWPPG ) ;
         end ;            (* end if *)
      GET (PPG) ;
    end;                  (* end while *)
    SETPAGE;
    for  COL :=  1      to  MAXCOL      do
```

```
          begin
             SEEK (NEWFORM, COL - 1) ;
             GET (NEWFORM) ;
             DICTIONARY ;
             INDEXFILE ;
          end ;
       CLOSE (NEWFORM, LOCK) ;
       CLOSE (PPG,LOCK ) ;
       CLOSE (FORM, LOCK ) ;
    end;  (* GETPAGEFORM *)



procedure  DFIRST ;
var

  MAXCNT        : integer ;
  SP            :string   ;

 procedure GETCHAP ;(********** G E T C H A P ***********)

   begin
      PAGE (OUTPUT );
      BOOK := NEWLIB^.TITLES [ BI ] ;
      RESET (CONTENTS, CONCAT (PREFIX, BOOK, '.BOOK' ) );
      if  CONTENTS^.NCHAP   =  0   then
         begin
            NEWCONT^ := CONTENTS^ ;
            FILER :=CONCAT (NDRIVE,BOOK, '.BOOK');
      REWRITE (NEWCONT, FILER ) ;
      CLOSE (NEWCONT, LOCK ) ;
      EXIT ( GETCHAP) ;
   end ;
WRITELN  ;
WRITELN ('chapters in ' , BOOK , '.BOOK' );
for CCNT := 1    to    CONTENTS^.NCHAP    do
   begin
      GOTOXY (LEFT, 1 + CCNT );
      FILER := CONTENTS^.HEADING [ CCNT ] ;
      SP := COPY (BLANK, 1 , (MAXCHAP - LENGTH (FILER) ) );
      WRITELN ( CCNT , '    ', FILER ,SP, '====>' );
      WORKQUE.TITLE [ CCNT ] := FILER ;
      WORKQUE.PSW [ CCNT ] := CONTENTS^.PASWORD [ CCNT ] ;
   end;
   WORKQUE.IT := CONTENTS^.NCHAP ;
   MAXCNT := CONTENTS^.NCHAP;
   ACCEPT ( WORKQUE, MAXCNT)   ;
   PAGE ( OUTPUT );
   if WORKQUE.IT = 0  then
     begin
       CLOSE ( CONTENTS, LOCK) ;
       EXIT ( GETCHAP) ;
     end ;
   DUMMYJOB := FALSE ;
   FOOTAREA ;
   WRITELN ('** BACKUP : ' , BOOK , '.BOOK') ;
   FILER :=CONCAT (NDRIVE,BOOK, '.BOOK');
   REWRITE (NEWCONT, FILER );
   with  NEWCONT^  do
```

```
        begin
           for   CCNT :=  1    to  WORKQUE.IT      do
              begin
                 HEADING [ CCNT ] := WORKQUE.TITLE [ CCNT ] ;
                 PASWORD [ CCNT ] := WORKQUE.PSW [ CCNT ] ;
              end ;
           NCHAP := WORKQUE.IT ;
           if  NCHAP <  NCHAPS    then
              begin
                 for   CCNT := NCHAP + 1   to  NCHAPS   do
                    begin
                       HEADING [ CCNT ] := COPY ( BLANK, 1, MAXCHAP);
                       PASWORD [ CCNT ] := COPY ( BLANK, 1, MAXCHAP);
                    end;

               end;
           end;
           SEEK (NEWCONT, 0);
           PUT (NEWCONT );
           CLOSE ( NEWCONT , LOCK ) ;
           for  CI := FIRST        to  NEWCONT^.NCHAP  do
                begin
                   GETPAGEFORM;
                   CLOSE (CONTENTS,LOCK );
               end ;
        end;                         (* GETCHAP *)

(********************** D F I R S T **********************)
```

```
begin
   PAGE (OUTPUT);
   FILER := CONCAT ( LIBNAME, '.LIB');
   RESET (LIBRARY,FILER) ;
   if  LIBRARY^.ITEMS  =  0  then
      begin
         NEWLIB^ := LIBRARY^ ;
         FILER := CONCAT ( NDRIVE , 'MICROBOOK.LIB' ) ;
         REWRITE (NEWLIB, FILER ) ;
         CLOSE (NEWLIB, LOCK ) ;
         EXIT ( DFIRST ) ;
      end ;
   WRITELN ;
   WRITELN ('Books in ', 'MICROBOOK.LIB', ' :');
   for  BCNT := 1  to LIBRARY^.ITEMS   do
      begin
         GOTOXY (LEFT, 1 + BCNT );
         FILER := LIBRARY^.TITLES [ BCNT ] ;
         SP := COPY (BLANK, 1 , ( MAXCHAP - LENGTH (FILER) ) );
         WRITELN (BCNT  , '     ' , FILER, SP, '===>');
         WORKQUE.TITLE [ BCNT ] := FILER ;
         WORKQUE.PSW [ BCNT ]    := LIBRARY^.PASSWORD [ BCNT ];
      end;
   WORKQUE.IT := LIBRARY^.ITEMS ;
   MAXCNT := LIBRARY^.ITEMS ;
   ACCEPT (WORKQUE, MAXCNT) ;
   PAGE ( OUTPUT ) ;
   if  WORKQUE.IT = 0    then
      EXIT (DFIRST) ;
   FOOTAREA ;
```

```
  FILER := CONCAT ( NDRIVE , 'MICROBOOK.LIB' ) ;
  WRITELN ('** BACKUP :  MICROBOOK.LIB ') ;
  REWRITE (NEWLIB, FILER);
  for BCNT := 1 TO WORKQUE.IT    do
     begin
        NEWLIB^.TITLES [ BCNT ] := WORKQUE.TITLE [ BCNT ];
        NEWLIB^.PASSWORD [ BCNT ] := WORKQUE.PSW [ BCNT ];
     end ;
  NEWLIB^.ITEMS := WORKQUE.IT ;
  if  NEWLIB^.ITEMS < NBOOKS    then
     begin
        with  NEWLIB^  do
           begin
              for  BCNT := ITEMS + 1   to  NBOOKS   do
                 begin
                    TITLES [ BCNT ] := SHORT ;
                    PASSWORD [ BCNT ] := SHORT ;
                 end; (* end with *)
            end;
      end;            (*  end if *)
   SEEK (NEWLIB, 0);
   PUT (NEWLIB);
   RESET (NEWDICT, CONCAT(NDRIVE, 'MICROBOOK.DICT'));
   RESET (NEWAVAIL,CONCAT(NDRIVE,'MICROBOOK.AVAIL'));
     for  BI := FIRST     TO  NEWLIB^.ITEMS      do
           GETCHAP;
end;           (* end of DFIRST      *)
```

```
procedure  SETDICT ;
   begin
     FOOTAREA ;
     WRITELN ('** BACKUP :  MICROBOOK.DICT ') ;
     FILER := CONCAT (NDRIVE,'MICROBOOK.DICT');
     REWRITE (NEWDICT, FILER);
     WORSIZE := NODECOUNT ( VWORDS, NBTREE - 1  ) ;
     NEWDICT^.FLAG := false ;
     NEWDICT^.LINK := NULL  ;
     for ITEM := 1     TO   NBTREE   do
        begin

              NEWDICT^.POINTER [ ITEM ] := NULL ;
              with  NEWDICT^.SUBKEY [ ITEM ]   do
                 begin
                    WKEY := COPY (BLANK, 1, MAXKEY) ;
                    TITLE := COPY (BLANK,1,MAXCHAP) ;
                    CHAP := COPY (BLANK, 1, MAXCHAP) ;
                    COLMN := YORIGIN ;
                    XPOS := XORIGIN ;
                    YPOS := YORIGIN ;
                    LONG := XORIGIN;
                    PLACE := 0 ;
                 end;
        end;                                         (* end of with *)
     for  TSIZE := 0    to  WORSIZE        do
        begin
           SEEK  (NEWDICT,TSIZE);
           PUT   (NEWDICT);
        end;
   BLANKWORDS := NEWDICT^.SUBKEY [ NBTREE ] ;     (* for SQUEEZE *)
   CLOSE (NEWDICT,LOCK);

end ;                                  (* SETDICT *)


(*************************** B  A C K U P  M A I N B O D Y *******)

begin
   PAGE (OUTPUT);
   WRITE ('B A C K U P    M I C R O B O O K : ');
   GOTOXY (LEFT ,6);
   WRITE ( 'Insert destination disk, strike <rtn>');
   READ (KEYIN);
   PAGE (OUTPUT) ;
   PROMPTS (LEFT, UPPER, 'Enter volume number or name ',
                  NDRIVE ) ;
   repeat
      SHOW (UPPER,UPPER,UPPER) ;
      WRITE ('Type max # keywords in dictionary ');
      READLN (VWORDS);
      SHOW (UPPER,UPPER,UPPER) ;
      WRITE ('Are you sure ? ');
      READ (KEYIN);
   until  KEYIN = 'Y';
   SETDICT ;
   SHOW (UPPER,UPPER,UPPER) ;
   WRITELN (WORSIZE, '  Records allocated for dictionary ');
   FILER := CONCAT ( NDRIVE,'MICROBOOK.AVAIL');
```

```
      REWRITE (NEWAVAIL, FILER ) ;
      with  NEWAVAIL^   do
         begin
            TREESIZE := WORSIZE ;
            TN := 0 ;
            ROOT := 0;
         end;
      PUT (NEWAVAIL);
      CLOSE (NEWAVAIL,LOCK);
      DUMMYJOB := TRUE ;
      DFIRST;
      CLOSE ( LIBRARY, LOCK ) ;
      CLOSE (NEWLIB,LOCK) ;
      if  not DUMMYJOB    then
           UPDICT
      else
         begin
            PAGE(OUTPUT) ;
            GOTOXY ( 5 , 12) ;
            WRITELN ('******* All books empty *******') ;
            READ (KEYIN) ;
            CLOSE ( NEWAVAIL, LOCK) ;
            CLOSE ( NEWDICT, LOCK) ;
         end ; (* else *)
end ;  (* BACKUP *)
```

```
procedure FILLFORM ;

 var
  ITEM, IT : integer ;
  FOUND    : boolean ;
  DATAWORD : string[ MAXWORD ] ;
  OPENFILE : string[ MAXCHAP ] ;
  COL      : integer ;

 procedure OK ;
  begin
    PROMPTS( LEFT, UPPER,
    'Enter N(ew file , A(dd to file : ', LINEIN ) ;
    if COPY( LINEIN, 1, 1 ) = 'A'
      then KEYIN := 'A'
      else KEYIN := 'N' ;
    case KEYIN of
     'N' : begin
            IT := 0 ;
            RESET ( PPG, CONCAT( PREFIX, CHAPHEAD, '.PAGES' ) ) ;
           end ;
     'A' : begin
            RESET( PPG, CONCAT( PREFIX, CHAPHEAD, '.PAGES' ) ) ;
            RESET( FORM, CONCAT( PREFIX, CHAPHEAD, '.FORM' ) ) ;
            while ( not EOF( PPG ) ) and ( PPG^.DPAGENO <> NULL )
                  do
                    begin
                     IT := PPG^.DPAGENO ;
                     GET( PPG ) ;
                     IT := IT + 1 ;
                     if IT > FORM^.MAXPAGE
                      then
                       begin
                        GOTOXY (XORIGIN, XORIGIN) ;
                        WRITE ('Chapter overflow. Strike <return>.') ;
                        READ (KEYIN) ;
                        CLOSE (PPG, LOCK) ;
                        CLOSE (CONTENTS, LOCK) ;
                        CLOSE ( FORM, LOCK ) ;
                        EXIT (FILLFORM) ;
                       end ;
                end ;
           end ;
    end ;
    GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
    WRITE ('Strike <rtn> to enter page = ', IT );
    READ( KEYIN ) ;
 end ;  (* OK  *)

function OPENCHAP : boolean ;     (* open chapter form and passwords *)


function  RUN : boolean ;
 begin
  RUN := TRUE ;  PAGE( OUTPUT ) ;
  PROMPTS( LEFT, UPPER, 'Enter title of book : ', LINEIN ) ;
  BOOK := LINEIN ;
  RESET( LIBRARY, CONCAT( LIBNAME, '.LIB' ) ) ;
  with LIBRARY^ do
   begin
   FOUND := FALSE ;
   for ITEM := 1 to ITEMS  do
    if TITLES[ ITEM ] = BOOK
     then begin FOUND := TRUE ; IT := ITEM end; (* for loop *)
   if FOUND
    then
     begin
      GOTOXY( LEFT, UPPER) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
      WRITE(  'Enter book password : '); READLN( KEYBOARD, LINEIN ) ;
      if PASSWORD[ IT ] <> LINEIN
       then
        begin
         GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
         WRITE( 'Wrong book password. Strike <return>. ' ) ;
         READ( KEYIN ) ;
         CLOSE( LIBRARY, LOCK ) ;
         RUN := FALSE ;
        end ; (* if *)
     end  (* then *)
    else
     begin
      GOTOXY( LEFT, UPPER) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
      WRITE( 'Book not found. Strike <return>. ' ) ;
      READ( KEYIN ) ;
      CLOSE( LIBRARY, LOCK ) ;
      RUN := FALSE ;
     end ; (* else *)
  (*  BOOK AND PASSWORD O.K. *)
  end (* with *);
end ;     (* RUN *)
```

```
begin
  OPENCHAP := TRUE ;
  PAGE( OUTPUT ) ;
  if not RUN then begin OPENCHAP := FALSE ; EXIT( OPENCHAP ) ; end ;
  CLOSE( LIBRARY, LOCK ) ;
  RESET( CONTENTS, CONCAT( PREFIX, BOOK,'.BOOK' ) ) ;
  SEEK( CONTENTS, 0 ) ; GET( CONTENTS ) ;
  GOTOXY( 5,3 ) ;
  WRITE( CONTENTS^.NCHAP, ' Chapters in ', BOOK ) ;
  for ITEM := 1 to CONTENTS^.NCHAP do
   begin
    GOTOXY( 5, 3+ITEM ) ;
    WRITE( 'Chapter [', ITEM, '] = ', CONTENTS^.HEADING[ ITEM ] ) ;
  end ;
  PROMPTS( LEFT, UPPER, 'Enter chapter heading : ', LINEIN ) ;
  CHAPHEAD := LINEIN ;
  with CONTENTS^  do
   begin
    FOUND := FALSE ;
    for ITEM := 1 to NCHAP do
     if HEADING[ ITEM ] = CHAPHEAD
      then begin FOUND := TRUE; IT := ITEM end ;  (* for loop *)
    GOTOXY( LEFT, UPPER) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
    WRITE( 'Enter chapter password : ' ) ;
    READLN( KEYBOARD, PW ) ;
    if ( PASWORD[ IT ] <> PW ) or ( not FOUND )
        then
         begin
          CLOSE( CONTENTS, LOCK ) ;
          OPENCHAP := FALSE ;
```

```
           GOTOXY( LEFT, UPPER); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
           if FOUND
            then WRITE( 'Wrong chapter password. Strike <rtn>. ' )
            else WRITE( 'Book not found. Strike <rtn>.' ) ;
           READ( KEYIN ) ;
           EXIT( OPENCHAP ) ;
          end ;  (* if   *)
    end ; (*  with   *)
   OK ;
end ;  (*   OPENCHAP   *)

procedure GETFORM( COL : integer ) ;
 begin
  SEEK( FORM, COL - 1 );
  GET( FORM ) ; GOTOXY( LEFT, YORIGIN ) ;
  with  FORM^ do
   begin
    for ITEM := YORIGIN to NLINES do
     begin
      if POS( '/', PROMPT[ ITEM ] ) = 0
       then WRITELN( PROMPT[ ITEM ] );
    end ; (* for  *)
   GOTOXY( LEFT, UPPER ); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
   if FORM^.NCOL > 1 then
      begin
        WRITE( 'Strike <rtn> when ready to enter column = ', COL ) ;
        READ( KEYIN ) ;
     end ;
  end; (*  with  *)
end ;   (* GETFORM  *)
```

```
procedure GETWORDS( COL : integer ) ;
 var
  I : integer ;
  LINE : string ;
  NOLINE : boolean ;
 begin
  ITEM := 0 ;
  for I := YORIGIN to FORM^.NLINES do
   begin
    LINE :=CONCAT( FORM^.PROMPT[ I ], '.' ) ;
    NOLINE := POS( '/', LINE ) = 1 ;
    repeat
     if POS( '-', LINE ) > 0
      then
       begin
        DELETE( LINE, 1, POS( '-', LINE ) ) ;
        ITEM := ITEM + 1 ;
        while POS( '-', LINE ) = 1 do DELETE( LINE, 1, 1 ) ;
        if not NOLINE
         then
          begin
           with FORM^.ENTRY[ ITEM ] do
            begin
             GOTOXY( XPOS, YPOS ) ; READLN( LINEIN ) ;
             if LENGTH( LINEIN ) > LONG
              then DATAWORD := COPY( LINEIN, 1, LONG )
              else DATAWORD := LINEIN ;
            with PPG^ do
             begin
              VALUES[ ITEM ] := DATAWORD;
              DPAGENO := IT ; DCOL := COL ;
             end ;  (*  with  *)
            end ;  (*  with  *)
           end ;  (* if not NOLINE  *)
         end ; (*  if POS('-',LINE) then  *)
    until POS( '-', LINE ) = 0 ;
   end ;  (*  for I  *)
 end ;  (* GETWORDS  *)

procedure NEWORD ( I : integer ) ;
 begin
  with FORM^.ENTRY[ I ] do
   begin
    GOTOXY( XPOS, YPOS ) ; WRITE( COPY( BLANK, 1, LONG ) ) ;
    GOTOXY( XPOS, YPOS ) ; READLN( LINEIN ) ;
    if LENGTH( LINEIN ) > LONG
     then DATAWORD := COPY( LINEIN, 1, LONG )
     else DATAWORD := LINEIN ;
    PPG^.VALUES[ I ] := DATAWORD ;
   end ;   (*  with  *)
 end ;   (*   NEWORDS  *)

 procedure FINISH ;
   begin
     SEEK ( AINDEX, 0 ) ;
     PUT ( AINDEX ) ;
     CLOSE( AINDEX, LOCK ) ;
     CLOSE( PINDEX, LOCK ) ;
   end ;                        (* FINISH *)
```

```
procedure INDEXWORD( DWORD : PLOCATION ; BTREE : FNAMES );
  (*$G+ *)
    label
      4 ;

    const
     NULL = -1 ;

    var
      I, K, LAST : integer ;          (* WORKING COUNTERS, ETC            *)
      NEW, TEMP   : PAGEDATA   ;         (* WORKING NODES                    *)
      NEWNODE,
      NODE,
      TEMPPTR     : integer  ;      (* WORKING POINTERS                 *)
      STOP        : -1 .. 20 ;      (* STACK PPOINTER                    *)
      STACK       : array[ 0..20 ] of integer ;
      KEY         : PLOCATION;      (* THE SEARCH KEY                   *)
      ITEM        : integer  ;      (* THE POINTER TO BTREE NODES FILE*)

 function ALOCATE : integer ;
   begin
     with AINDEX^ do
      begin
        PN := PN + 1 ;
        ALOCATE := PN;
      end  (*  with  *)
 end;                        (*  ALOCATE    *)

 function  LOOKUP : boolean ;
   begin
  LOOKUP := TRUE ;
  STOP := 0 ; STACK[ STOP ] := NULL ;
  STOP := 1 ; STACK[ STOP ] := NULL ;
  NODE := AINDEX^.PROOT ;       (* START AT PROOT NODE        *)
  repeat
    I := 1 ;
    SEEK( PINDEX, NODE ) ;
    GET ( PINDEX ) ;
    with PINDEX^ do
      begin
        while ( PPSUBKEY[ I ].PKEY <> COPY( BLANK, 1, MAXWORD )  )
                    and
              ( PPSUBKEY[ I ].PKEY < KEY.PKEY )
         do
          begin
          I := SUCC( I ) ;       (* FIND INSERT PLACE          *)
          end ;
        STOP := STOP + 1 ; STACK[ STOP ] := NODE ;
        STOP := STOP + 1 ; STACK[ STOP ] := I ;
        NODE := PPOINTER[ I ] ;
        if PPSUBKEY[ I ].PKEY = KEY.PKEY
           then
            begin
             LOOKUP := FALSE ;
            end ;
      end ;                           (* with *)
  until ( NODE = NULL ) or ( PINDEX^.PFLAG );
end ;                                 (*  LOOKUP  *)
```

```
procedure SPLIT ;
  begin
    with  PINDEX^ do
      begin
        NEW.PLINK := PLINK ;
        NEW.PFLAG := PFLAG ;
        LAST     := (NPP + 1) div 2 ;
        K        := 1 ;
        while LAST+K <= NPP do
          begin
            NEW.PPOINTER[ K ] := PPOINTER[ LAST + K ] ;
            NEW.PPSUBKEY [ K ] := PPSUBKEY [ LAST + K ] ;
            K := SUCC ( K ) ;
          end ;               (* while *)
        NEW.PPOINTER[ K ] := TEMPPTR ;
        NEW.PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
        for K := K+1 to NPP do
          begin
            NEW.PPOINTER[ K ] := NULL ;
            NEW.PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
          end ;               (* while *)
      end ;                (* with  *)

   (*   STEP   7   *)

   TEMP    := PINDEX^ ;
   PINDEX^  := NEW ;
   NEWNODE := ALOCATE ;
   SEEK( PINDEX, NEWNODE ) ;
   PUT ( PINDEX ) ;
```

```
     PINDEX^  := TEMP ;
     with PINDEX^ do
       begin
         KEY.PKEY   := PPSUBKEY[ LAST ].PKEY ;
         for K := LAST+1  to NPP do
           begin
             PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
             PPOINTER[ K ] := NULL ;
           end ;                          (*  for  *)
         PLINK := NEWNODE ;
       end ;                  (* with *)
     SEEK( PINDEX, NODE ) ;
     PUT ( PINDEX ) ;
end ;                   (* SPLIT *)

  procedure NEWROOT ;
    begin
      with  PINDEX^ do
        begin
          PFLAG := FALSE ;
          PPOINTER[ 1 ] := ITEM ;
          PPSUBKEY [ 1 ] := KEY ;
          PPOINTER[ 2 ] := NEWNODE ;
          PPSUBKEY [ 2 ].PKEY := COPY( BLANK, 1, MAXWORD );
          for K := 3 to NPP do
            begin
              PPOINTER[ K ] := NULL ;
              PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
            end ;
          PLINK := NULL ;
          NEWNODE := ALOCATE ;
        end ;           (* with *)
        SEEK( PINDEX, NEWNODE ) ;
        PUT ( PINDEX ) ;
        AINDEX^.PROOT := NEWNODE ;
   end  (*  NEWROOT  *) ;

    (******    BODY OF INDEXWORD     *****)

begin
 RESET( PINDEX,CONCAT( PREFIX, BTREE, '.INDEX' ) ) ;
 RESET( AINDEX,CONCAT( PREFIX, BTREE, '.AVAIL' ) ) ;
 GOTOXY( LEFT, UPPER ); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
 WRITE( 'Indexing : ', DWORD.PKEY, ' in file : ', BTREE, '.INDEX' ) ;
 KEY := DWORD ;
 ITEM := NULL ;
 if not LOOKUP then WRITE(' Duplicate' ) ;
 I         := STACK[ STOP ] ; STOP := STOP - 1 ;
 NODE      := STACK[ STOP ] ; STOP := STOP - 1 ;
 NEW.PFLAG := TRUE ;

  (*     STEP   4      *)

  4:
   with PINDEX^ do
     begin
       TEMPPTR := PPOINTER[ NPP ] ;
       for K := NPP downto I + 1 do
         begin
           PPOINTER[ K ] := PPOINTER[ K - 1 ] ;
```

```
      PPSUBKEY [ K ] := PPSUBKEY [ K - 1 ] ;
    end ;
  PPOINTER[ I ] := ITEM ;
  PPSUBKEY [ I ] := KEY ;
  PFLAG          := NEW.PFLAG ;
end ;            (*  with  *)

(*   STEP   5    *)

if PINDEX^.PPSUBKEY[ NPP ].PKEY = COPY( BLANK, 1, MAXWORD )
  then
    begin
      SEEK( PINDEX, NODE ) ;
      PUT ( PINDEX ) ;
      FINISH ;
      EXIT( INDEXWORD ) ;
    end ;

SPLIT ;

ITEM   := NODE ;
I      := STACK[ STOP ] ; STOP := STOP - 1 ;
NODE   := STACK[ STOP ] ; STOP := STOP - 1 ;
if NODE = NULL
  then
    begin
    NEWROOT ;
    FINISH ;
    EXIT( INDEXWORD );
  end ;     (* IF-THEN  *)
  SEEK( PINDEX, NODE ) ;
  GET ( PINDEX ) ;
  PINDEX^.PPOINTER[ I ] := NEWNODE ;
  NEW.PFLAG := FALSE ;
  goto 4 ;

end;                        (* INDEXWORD *)

 procedure PUTDATA ;
  var
   BTREE   : FNAMES ;
   DATAWORD : PLOCATION ;

  begin
   with FORM^ do
    for ITEM := 1 to NENTRIES do
     begin
      GOTOXY( ENTRY[ ITEM ].XPOS, ENTRY[ ITEM ].YPOS ) ;
      WRITE( COPY( BLANK, 1, ENTRY[ ITEM ].LONG ) ) ;
     if POS( '*', ENTRY[ ITEM ].WKEY ) = 1
      then
       begin
        BTREE := COPY( ENTRY[ ITEM ].WKEY, 2, LENGTH( ENTRY[ ITEM ].WKEY)-1);
        with DATAWORD do
         begin
          PKEY := PPG^.VALUES[ ITEM ] ;
          PTITLE := BOOK ;
          PHEAD  := CHAPHEAD ;
          PP  := IT ;
          PCOLUMN := COL ;
          INDEXWORD( DATAWORD, BTREE ) ;
         end ;
       end ;  (* if  *)
     end ;  (*  for  *)
    SEEK (PPG, (IT*FORM^.NCOL + COL -1) ) ;
    PUT( PPG ) ;        (* STORE DATA WORD IN PAGE FILE  *)
    GOTOXY( LEFT, UPPER ); WRITE( BLANK ); GOTOXY( LEFT, UPPER );
    WRITE( 'Page ', IT, ' entered. Strike <rtn>.' ) ;
    READ( KEYIN ) ;
  end ;  (* putdata  *)

begin      (* F I L L F O R M    B O D Y   *)
    PAGE( OUTPUT ) ;
    OPENFILE := COPY( BLANK, 1, MAXCHAP ) ;
    if not OPENCHAP then EXIT( FILLFORM ) ;
    CLOSE( FORM ) ;
    RESET( FORM, CONCAT( PREFIX, CHAPHEAD, '.FORM' ) ) ;
    repeat    (* enter as many pages as you want  *)
     for COL := 1 to FORM^.NCOL do
      begin
       PAGE( OUTPUT ) ;
       GETFORM( COL ) ;    (* DISPLAY FORM ON SCREEN *)
       GETWORDS( COL );    (* GET DATA VALUES FROM USER *)
       repeat
        GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
        WRITE( 'Enter number to change, 0 to stop :' ) ;
        READLN( ITEM ) ;
        if ( ITEM <> 0 ) and ( ITEM <= FORM^.NENTRIES ) then NEWORD( ITEM ) ;
       until ITEM = 0 ;
       PUTDATA ;          (* WRITE DATA TO DISK   *)
      end ;  (* for COL *)
      IT := IT + 1 ;       (* NEXT PAGE  *)
      GOTOXY ( LEFT, UPPER ) ;WRITE('Enter S(top, A(dd more pages : ');
      READ( KEYIN ) ;
      if KEYIN = 'A' then
           if IT > FORM^.MAXPAGE
              then
               begin
                WRITE ('Chapter overflow. Strike <return>.') ;
                READ (KEYIN) ;
                KEYIN := 'S' ;
               end ;
     until KEYIN = 'S' ;
     CLOSE( CONTENTS, LOCK ) ;
     CLOSE( FORM, LOCK ) ;
     CLOSE( PPG, CRUNCH ) ;   (* POINT TO LAST RECORD IN FILE *)
   end ;  (*   FILLFORM  *)


PROGRAM  CHANSYS ;
  const  NCONST = 9 ;
  type   CNAME  = string[10] ;
  var    I, CTYPE  : integer ;
         KEYIN     : char ;
         X         : TEXT ;      (* FILE OF CHAR *)
         LINE      : string[80] ;
         NVALUE    : string[ 3] ;
         NAME      : array [1..NCONST] of CNAME ;
         VALUE     : array [1..NCONST] of string[3] ;
  PROCEDURE  OLDCOPY ;
    var   I : integer ;
    begin
       RESET(X, ’SOURCE:ODATA.TEXT’) ;
       for I:=1 to NCONST do
         begin
           readln( X ) ;
           write( X, CONCAT( ’O’,NAME[I], VALUE[I] ) )
         end ;
       CLOSE( X, LOCK ) ;
    end ;            (* end OF OLDCOPY *)


begin
   PAGE( OUTPUT ) ;
   writeln ;
   writeln(’ Remove  MODIFY:’) ;
   writeln(’ Put in  SOURCE:’) ;
   writeln(’ Strike <return> key.’) ;
   read( KEYIN ) ;
   PAGE( OUTPUT ) ;
   writeln(’  CURRENT SYSTEM: ’) ;
   writeln ;
   writeln(’    type    name      value   ’) ;
   RESET( X, ’SOURCE:DATA.TEXT’) ;
   readln( X ) ;
   for  I:=1  to  NCONST  do
     begin
       readln( X, LINE ) ;
       NAME[ I ] := COPY( LINE, 4, 10 ) ;
    VALUE[ I ] := COPY( LINE, 14, 3 ) ;
    writeln('     ',I:2,'      ',LINE) ;
  end ;
CLOSE( X, LOCK ) ;
writeln ;
write( ' New system change?(Y/N) ') ;
readln( KEYIN ) ;
if  KEYIN = 'N'  then
   begin
      OLDCOPY ;
      KEYIN := 'Y'
   end ;


while  KEYIN = 'Y'  do
   begin
      writeln ;
      write(' Enter type number: ') ;
      readln( CTYPE ) ;
      write(' Enter new value(3-digit): ') ;
      readln( NVALUE ) ;
      RESET( X, 'SOURCE:DATA.TEXT' ) ;
      for I:=1 to CTYPE do readln( X ) ;
      write( X, CONCAT( NAME[CTYPE], NVALUE ) ) ;
      CLOSE( X, LOCK ) ;
      RESET( X, 'UNIT:UNIT.TEXT' ) ;
      for I:=1 to CTYPE+6 do readln( X ) ;
      write( X, CONCAT( NAME[CTYPE], NVALUE ) ) ;
      CLOSE( X, LOCK ) ;
      write(' Change more?(Y/N) ') ;
```

```
         readln( KEYIN )
      end ;
   PAGE( OUTPUT ) ;
   writeln ;
   writeln( ' Remove  SOURCE:' ) ;
   writeln( ' Put in  MODIFY:' ) ;
   writeln( ' Strike <return> key.' ) ;
   readln( KEYIN ) ;
   PAGE( OUTPUT ) ;
   writeln ;  writeln ;
   writeln( ' Enter  " X  EXEC/UNIT:CHANSYS1 "  for the first step.' ) ;
end .


PROGRAM  CONVERT ;

  (*$S+*)
 var  I, IDLE : integer ;

      (*$I SOURCE:DATA*)
      (*$I SOURCE:ODATA*)

  PROCEDURE  LIBFILE ;
  var  I        : integer ;
       SHORT    : FNAMES ;

  begin
     RESET( OLIBRARY, CONCAT( '#4:', LIBNAME, '.LIB') ) ;
     REWRITE( LIBRARY, CONCAT( '#5:', LIBNAME, '.LIB') ) ;
     SHORT := COPY( BLANK, 1, MAXCHAP ) ;
   with  LIBRARY^  do
     with  OLIBRARY^  do
       begin
         ITEMS := OITEMS ;
         for I:=1 to ITEMS do
           begin
             TITLES[I]    := OTITLES[I] ;
             PASSWORD[I]  := OPASSWORD[I] ;
             BOPENED[I]   := OBOPENED[I] ;
           end ;
         for I:=ITEMS+1 to NBOOKS do
           begin
             TITLES[I]    := SHORT ;
             PASSWORD[I]  := SHORT
           end ;
       end ;        (* with *)
   PUT( LIBRARY ) ;
   CLOSE( LIBRARY, LOCK ) ;
   CLOSE( OLIBRARY, LOCK ) ;
end ;                                   (*   LIB  F I L E  *)

PROCEDURE  DICTFILE ;
var   I, ITEM, SIZE, AVAIL, CNODE : integer ;
      BKEY                        : WORDS ;
      SBLANK                      : string[OMAXKEY] ;

begin
   RESET( OAVAILIST, CONCAT( '#4:', LIBNAME, '.AVAIL') ) ;
   REWRITE( AVAILIST, CONCAT( '#5:', LIBNAME, '.AVAIL') ) ;
   with  AVAILIST^  do
```

```
  begin
    TREESIZE := OAVAILIST^.OTREESIZE ;
    SIZE     := TREESIZE ;
    TN       := OAVAILIST^.OTN ;
    AVAIL    := TN ;
    ROOT     := OAVAILIST^.OROOT
  end ;      (* with *)                    (* COPY OF AVAIL-FILE *)
PUT( AVAILIST ) ;
CLOSE( AVAILIST, LOCK ) ;
RESET( ODICTION, CONCAT( '#4:', LIBNAME, '.DICT') ) ;
REWRITE( DICTION, CONCAT( '#5:', LIBNAME, '.DICT') ) ;
SBLANK := COPY( BLANK, 1, OMAXKEY ) ;
with  BKEY  do
  begin
    WKEY  := COPY( BLANK, 1, MAXKEY) ;
    TITLE := COPY( BLANK, 1, MAXCHAP ) ;
    CHAP  := COPY( BLANK, 1, MAXCHAP ) ;
    COLMN := 0 ;
    XPOS  := XORIGIN; YPOS := YORIGIN; LONG := XORIGIN;
    PLACE := 0
  end ;        (* with *)
for  CNODE:=0  to  AVAIL  do
  begin
    with  DICTION^  do
      begin
        with  ODICTION^  do
          begin
            FLAG := OFLAG ;
            LINK := OLINK ;
            ITEM := 1 ;
      repeat
        POINTER[ ITEM ] := OPOINTER[ ITEM ] ;
        with  SUBKEY[ITEM]  do
          with  OSUBKEY[ITEM]  do
            begin
              WKEY  := OWKEY ;
              TITLE := OTITLE ;
              CHAP  := OCHAP ;
              COLMN := OCOLMN ;
              XPOS  := OXPOS;  YPOS := OYPOS;
              LONG  := OLONG ;
              PLACE := OPLACE ;
            end ;
        ITEM := ITEM+1 ;
        until OSUBKEY[ITEM].OWKEY = SBLANK ;
        POINTER[ ITEM ] := OPOINTER[ ITEM ] ;
        SUBKEY[ ITEM ]  := BKEY
      end ;                          (* with ODICTION *)
    for ITEM:=ITEM+1 to NBTREE do      (* FILL UP BLANK ENTRY *)
      begin
        POINTER[ ITEM ] := NULL ;
        SUBKEY[ ITEM ]  := BKEY
      end
  end ;                              (* with DICTION *)
  PUT( DICTION ) ;
  GET( ODICTION ) ;
end ;                                (* while *)
CLOSE( OAVAILIST, LOCK ) ;
CLOSE( ODICTION,  LOCK ) ;
for CNODE:= AVAIL+1 to SIZE do
```

```
    begin
      with  DICTION^  do
        begin
          FLAG := FALSE ;
          LINK := NULL ;
          for ITEM:=1 to NBTREE do
            begin
              POINTER[ ITEM ] := NULL ;
              SUBKEY[ ITEM]   := BKEY
            end
        end ;       (* with DICTION *)
      PUT( DICTION )
    end ;           (* for          *)
  CLOSE( DICTION, LOCK )
end ;                                  (* end OF D I C T  *)


PROCEDURE  BOOKFILE ;
var    I, CH  : integer ;
       BK     : FNAMES ;        (* BLANK *)

begin
  RESET( LIBRARY, CONCAT( '#5:', LIBNAME, '.LIB') ) ;
  BK := COPY( BLANK, 1, MAXCHAP ) ;
  for  I:=1  to  LIBRARY^.ITEMS  do
    begin
      RESET( OCONTENTS, CONCAT( '#4:', LIBRARY^.TITLES[I], '.BOOK') ) ;
      REWRITE( CONTENTS, CONCAT('#5:', LIBRARY^.TITLES[I], '.BOOK') ) ;
      with  CONTENTS^  do
        begin
```

```
          with  OCONTENTS^  do
            begin
              CH    := 1 ;
              NCHAP := ONCHAP ;
              if NCHAP > 0 then
                repeat
                  HEADING[ CH ] := OHEADING[ CH ] ;
                  PASWORD[ CH ] := OPASWORD[ CH ] ;
                  COPENED[ CH ] := OCOPENED[ CH ] ;
                  CH := CH + 1
                until ( CH > NCHAP )
            end ;                       (* with OCONTENTS *)
          for CH:=CH to NCHAPS do
            begin
              HEADING[ CH ] := BK ;
              PASWORD[ CH ] := BK
            end
        end ;                          (* with CONTENTS *)
      PUT( CONTENTS ) ;
      CLOSE( CONTENTS, LOCK ) ;
      CLOSE( OCONTENTS, LOCK )
    end ;                              (*  for  *)
  CLOSE( LIBRARY, LOCK )
end;                                   (*  end OF  B  O  O  K  *)


PROCEDURE  FORMMFILE ;
var   I, J, Y, CH,NBOOK, NCH, COLUMN, MAXCOL  : integer ;

begin
```

```
J := 1 ;
RESET( LIBRARY, CONCAT( '#5:', LIBNAME, '.LIB') ) ;
for  NBOOK:=1  to  LIBRARY^.ITEMS  do
  begin
    RESET( CONTENTS, CONCAT( '#5:', LIBRARY^.TITLES[NBOOK],'.BOOK') );
    NCH := CONTENTS^.NCHAP ;
    if  NCH <> 0 then
      for CH:=1 to NCH do
        begin
          RESET( OFORM, CONCAT( '#4:', CONTENTS^.HEADING[CH],'.FORM'));
          REWRITE( FORM, CONCAT( '#5:',CONTENTS^.HEADING[CH],'.FORM'));
          MAXCOL := OFORM^.ONCOL ;
          for COLUMN:=1 to MAXCOL do
            begin
              with  FORM^  do
                with  OFORM^  do
                  begin
                    NCOL    := ONCOL ;
                    MAXPAGE := OMAXPAGE ;
                    NLINES  := YLIMIT ;
                    NENTRIES:= ONENTRIES ;
                    for Y:=YORIGIN to YLIMIT do     (* COPY SCREEN *)
                      PROMPT[ Y ] := OPROMPT[ Y ] ;
                    for I:=1 to NENTRIES do
                      with ENTRY[I] do
                        with OENTRY[I] do
                          begin
                            WKEY  := OWKEY ;
                            TITLE := OTITLE ;
                            COLMN := OCOLMN ;
                            XPOS  := OXPOS ;
                            YPOS  := OYPOS ;
                            LONG  := OLONG ;
                            PLACE := OPLACE ;
                            if POS( '*', OWKEY ) = 1 then
                              begin
                     KEYWORD[J] := COPY(OWKEY,2,LENGTH(OWKEY)-1) ;
                                J := J + 1
                              end
                          end ;             (* with ENTRY *)
                  end ;                     (* with OFORM *)
              PUT( FORM ) ;
              GET( OFORM ) ;
            end ;                           (* for COLUMN *)
          CLOSE( FORM, LOCK ) ;
          CLOSE( OFORM, LOCK ) ;
        end ;                               (* for CH:=1 *)
      CLOSE( CONTENTS, LOCK )
    end ;
  KEYWORD[ J ] := '*' ;
  CLOSE( LIBRARY, LOCK )
end ;                                       (*  end OF  F O R M  *)


PROCEDURE  PPGFILE ;
var     I, PG, CH, NCH, NPAGE, NBOOK, MAX  : integer ;

begin
  RESET( LIBRARY, CONCAT( '#5:', LIBNAME, '.LIB' ) ) ;
  for  NBOOK:=1  to  LIBRARY^.ITEMS  do
```

```
begin
  RESET( CONTENTS, CONCAT( '#5:', LIBRARY^.TITLES[NBOOK], '.BOOK'));
  NCH := CONTENTS^.NCHAP ;
  if  NCH <> 0  then
    for CH:=1 to NCH do
      begin
        RESET( OFORM, CONCAT('#4:',CONTENTS^.HEADING[CH],'.FORM'));
        MAX := OFORM^.OMAXPAGE ;
        PG  := OFORM^.ONCOL ;
        CLOSE( OFORM, LOCK ) ;
        RESET( OPPG, CONCAT('#4:',CONTENTS^.HEADING[CH],'.PAGES'));
        REWRITE( PPG, CONCAT('#5:',CONTENTS^.HEADING[CH],'.PAGES'));
        NPAGE := 0 ;
        while (OPPG^.ODPAGENO = NPAGE) AND (NPAGE <= MAX) do
         begin
          for NCH:=1 to PG do
           begin
            with  PPG^  do
              with  OPPG^  do
                begin
                  DSTAT   := TRUE ;
                  DPAGENO := ODPAGENO ;
                  DCOL    := ODCOL ;
                  for I:=1 to ONWORDS do
                     VALUES[ I ] := OVALUES[ I ] ;
                end ;
            PUT( PPG ) ;
            GET( OPPG )
           end;
          NPAGE := NPAGE + 1
```

```
             end;                                (* while *)
            for I:=NPAGE to MAX do
             for NCH:=1 to PG do
              begin
                PPG^.DSTAT    := FALSE ;
                PPG^.DPAGENO := -1 ;
                PUT( PPG )
              end ;
            CLOSE( PPG, LOCK ) ;
            CLOSE( OPPG, LOCK )
          end ;                                    (* for CH:=1 *)
      CLOSE( CONTENTS, LOCK )
    end ;                                          (* for NBOOK *)
  CLOSE( LIBRARY, LOCK )
end ;                                              (*  end OF  P P G  *)


PROCEDURE  INDEXFILE ;
var     I, TEMP, NODE, ITEM   : integer ;
        BLOC                  : PLOCATION ;
        SBLANK                : string[ OMAXWORD ] ;

begin
  I := 1 ;
  while  KEYWORD[I] <> '*'  do
    begin
      RESET( OAINDEX, CONCAT( '#4:', KEYWORD[I], '.AVAIL' ) ) ;
      REWRITE( AINDEX, CONCAT( '#5:', KEYWORD[I], '.AVAIL' ) ) ;
      with  AINDEX^  do
        begin
    PN    := OAINDEX^.OPN ;
    PROOT:= OAINDEX^.OPROOT ;
    NEX   := OAINDEX^.ONEX ;
    NXI   := OAINDEX^.ONXI ;
    TEMP  := PN
  end ;
PUT( AINDEX ) ;
CLOSE( AINDEX, LOCK ) ;
CLOSE( OAINDEX, LOCK ) ;
RESET( OPINDEX, CONCAT( '#4:', KEYWORD[I], '.INDEX' ) ) ;
REWRITE( PINDEX, CONCAT( '#5:', KEYWORD[I], '.INDEX' ) ) ;
SBLANK := COPY( BLANK, 1, OMAXWORD ) ;
with  BLOC  do
  begin
    PKEY    := COPY( BLANK, 1, MAXWORD ) ;
    PTITLE  := COPY( BLANK, 1, MAXCHAP ) ;
    PHEAD   := COPY( BLANK, 1, MAXCHAP ) ;
    PP      := 0 ;
    PCOLUMN:= 1 ;
  end ;                            (* with *)
for  NODE:=0  to  TEMP  do         (* TEMP=# OF NODES USED *)
  begin
    with  PINDEX^  do
      begin
        with  OPINDEX^  do
          begin
            PFLAG := OPFLAG ;
            PLINK := OPLINK ;
            ITEM  := 1 ;
            repeat
```

```
              PPOINTER[ ITEM ] := OPPOINTER[ ITEM ] ;
              with  PPSUBKEY[ ITEM ]  do
                with  OPPSUBKEY[ ITEM ]  do
                  begin
                    PKEY   := OPKEY ;
                    PTITLE := OPTITLE ;
                    PHEAD  := OPHEAD ;
                    PP     := OPP ;
                    PCOLUMN:= OPCOLUMN ;
                  end ;
              ITEM := ITEM +1 ;
            until  OPPSUBKEY[ITEM].OPKEY = SBLANK ;
            PPOINTER[ ITEM ] := OPPOINTER[ ITEM ] ;
            PPSUBKEY[ ITEM ] := BLOC
          end ;                       (* with OPINDEX *)
        for ITEM:=ITEM+1 to NPP do
          begin
            PPOINTER[ ITEM ] := NULL ;
            PPSUBKEY[ ITEM ] := BLOC
          end ;                       (* with PINDEX *)
      end ;
    PUT( PINDEX ) ;
    GET( OPINDEX )
  end ;                               (* for NODE:=1 *)
while  NOT EOF(OPINDEX)  do
  begin
    with  PINDEX^  do
      begin
        PFLAG := FALSE ;
        PLINK := NULL ;
```

```
                 for ITEM:=1 to NPP do
                   begin
                     PPOINTER[ ITEM ] := NULL ;
                     PPSUBKEY[ ITEM ] := BLOC
                   end
               end ;                (* with PINDEX *)
             PUT( PINDEX ) ;
             GET( OPINDEX )
           end ;                    (* while NOT   *)
         CLOSE( PINDEX, LOCK ) ;
         CLOSE( OPINDEX, LOCK ) ;
         I := I + 1
       end ;                        (* while *)
   end ;                              (* end OF  I N D E X  *)


begin             (***********  M A I N  ***********)
   LIBNAME := 'MICROBOOK' ;
   PAGE( OUTPUT ) ;
   writeln ;
   writeln(' Remove  MODIFY:  and put in old  DATA:') ;
   writeln ;
   writeln(' Remove  SOURCE:  and put in new  DATA:') ;
   I := 1 ;
   for IDLE:=1 to 10000 do I:=I*I*I*I*I ;
   LIBFILE ;   writeln('LIBFILE===============>done') ;
   DICTFILE ;  writeln('DICTFILE==============>done') ;
   BOOKFILE ;  writeln('BOOKFILE==============>done') ;
   FORMMFILE ; writeln('FORMFILE==============>done') ;
   PPGFILE ;   writeln('PPGFILE===============>done') ;
   INDEXFILE ; writeln('INDEXFILE=============>done') ;
   PAGE( OUTPUT ) ;
   writeln ;
   writeln(' Remove old  DATA:  and put in  MODIFY:') ;
   writeln ;
   writeln(' Remove new  DATA:  and put in  MICRO: ') ;
   I := 1 ;
   for IDLE:=1 to 10000 do I:=I*I*I*I*I ;
end.



const
 ONBOOKS    =008 ;                          (* MAX SIZE OF LIBRARY       *)
 ONCHAPS    =005 ;                          (* MAX SIZE OF BOOK          *)
 ONCOLS     =002 ;                          (* MAX NO. COLUMNS PER PAGE *)
 ONWORDS    =012 ;                          (* NO. ENTRIES/PAGE          *)
 OMAXWORD   =020 ;                          (* MAX NO. CHAR/DATA WORD    *)
 ONBTREE    =012 ;                          (* ORDER OF B TREE           *)
 ONPP       =009 ;                          (* ORDER  OF INDEXES         *)
 OMAXKEY    =020 ;
 OMAXCHAP   =020 ;
 OXORIGIN   =001 ;  OXLIMIT  =065 ;  (* SCREEN WIDTH               *)
 OYORIGIN   =002 ;  OYLIMIT  =016 ;  (* SCREEN LENGTH              *)
 STAR       =300 ;
type
 OXAXIS     = OXORIGIN .. OXLIMIT ;   OYAXIS = OYORIGIN .. OYLIMIT ;
 OFNAMES    = string[ OMAXCHAP ] ;   (* FILE NAMES ARE STRINGS *)
 OBOOKS     = packed record           (* LIST OF OBOOKS IN LIBR. *)
               OITEMS   : 0..ONBOOKS ;
               OTITLES : array[ 1.. ONBOOKS ] of OFNAMES ;
               OPASSWORD : array[ 1..ONBOOKS ] of OFNAMES ;
               OBOPENED : packed array[1..ONBOOKS] of boolean ;
             end ;
OCHAPTERS   = packed record           (* CHAPTER HEADINGS      *)
               ONCHAP   : 0..ONCHAPS ;
               OHEADING: array[ 1..ONCHAPS ] of OFNAMES ;
               OPASWORD: array[ 1..ONCHAPS ] of OFNAMES ;
               OCOPENED: packed array[ 1..ONCHAPS ] of boolean ;
             end ;
OWORDS      = packed record           (* format of data dictionary entry *)
               OWKEY    : string[ OMAXKEY ] ;
               OTITLE   : string[ OMAXCHAP ] ;
               OCHAP    : string[ OMAXCHAP ] ;
               OCOLMN   : 0..ONCOLS ;
               OXPOS    : OXAXIS ;
               OYPOS    : OYAXIS ;
               OLONG    : integer ;
               OPLACE   : integer ;
             end ;
OPAGES      = packed record            (* FORM AND BLANK SIZES         *)
               ONCOL    : 1..ONCOLS ;
               OMAXPAGE : integer ;   (* max. no. of pages. *)
               OPROMPT : array[ OYAXIS ] of string[ OXLIMIT ] ;
               ONLINES : OYORIGIN..OYLIMIT ;
               OENTRY   : array[ 1..ONWORDS ] of OWORDS ;
               ONENTRIES : 1..ONWORDS ;
             end ;
```

```
ODATUM     = packed record          (* entry in ppg file of data   *)
              ODSTAT   : boolean ;  (* TRUE=KEEP, FALSE=DELETE      *)
              ODPAGENO : integer ;  (* page number *)
              ODCOL    : integer ;  (* column number *)
              OVALUES  : array[ 1..ONWORDS ] of string[ OMAXWORD ] ;
            end ;
ONODES    = packed record           (* form of data dictionary nodes *)
             OFLAG : boolean ;(* TRUE=NODE CONTAINS DATA POINTERS*)
                            (* FALSE=NODE HAS INDEX POINTERS   *)
             OPOINTER : array[ 1..ONBTREE ] of integer  ;
             OSUBKEY  : array[ 1..ONBTREE ] of OWORDS ;
             OLINK    : integer ;
           end ;
OEMPTIES  = packed record      (* header of data dictionary file *)
             OTREESIZE : integer ;
             OTN       : integer ;    (* NUMBER OF ONODES USED    *)
             OROOT     : integer  ;
           end ;
OPPDATA   = string[ OMAXWORD ];  (* value stored in index file    *)
OPLOCATION= packed record      (* points to value in data page  *)
             OPKEY    : OPPDATA ;(* data value                   *)
             OPTITLE  : OFNAMES ;(* book title         *)
             OPHEAD   : OFNAMES ;(* chapter heading  *)
             OPP      : integer;(* page number       *)
             OPCOLUMN : 1..ONCOLS ; (* column number *)
           end ;
OPAGEDATA = packed record          (* node in index b-tree files    *)
             OPFLAG   : boolean;(* T=leaf, F=internal node        *)
             OPPSUBKEY: array[ 1..ONPP ] of OPLOCATION ;
             OPPOINTER: array[ 1..ONPP ] of integer ;
             OPLINK    : integer ;
           end ;
OPUSED    = packed record          (* list of available space of index*)
             OPN       : integer;(* number of pages used, so far.  *)
             OPROOT    : integer;(* current root of b-tree index   *)
             ONEX      : integer;(* next pointer    *)
             ONXI      : integer;(* next subscript of key   *)
           end ;

var
  OLIBRARY  : file of OBOOKS ; (* contains the library of books *)
  OCONTENTS : file of OCHAPTERS;(*table of contents of a book   *)
  OFORM     : file of OPAGES ; (* file containing form of chapt *)
  OAVAILIST : file of OEMPTIES;(* list of available space for...*)
  ODICTION  : file of ONODES ; (* index file = b-tree structure *)
  OPINDEX   : file of OPAGEDATA;(* index to pages of a book     *)
  OAINDEX   : file of OPUSED ; (* list of available space in index*)
  OPPG      : file of ODATUM ; (* data file containing values   *)
  KEYWORD   : array[1.. STAR] of STRING[ OMAXKEY ] ;
```

14

Program Listing for UNIT

14.1 OVERVIEW

The MICROBOOK unit which is compiled and stored in SYSTEM.LIBRARY is given in its Pascal source code form on disk volume UNIT:. The main program of this unit is contained in file UNIT:UNIT.TEXT. This program consists of an interface part and an implementation part. The interface part is simply a list of all data structures, functions, and procedures which are held in common while any using program is executing. Thus, when a program uses this unit, all global data structures found in SOURCE:DATA.TEXT are commonly available. Also, all functions and procedures defined in the interface part are available to the using program. Program QUERY is an example of a using program. See the next chapter.

The overall structure of UNIT:UNIT.TEXT is summarized below:

```
unit MICROBOOK ;
  Interface
    (* uses chainstuff for program chaining *)
    (* all data found in DATA.TEXT is global *)
    (* all callable routines are listed *)
  Implementation
    (* list of routines which can be called *)
  begin
    (* initially executed part *)
  end.
```

The unit runs as follows: All routines which are called by a using program are linked to the using program. As soon as the using program is started, the initialization section of UNIT: UNIT.TEXT is executed. This is the part between *begin-end* above.

Next, the using program begins to execute. Each call to a unit procedure or function is handled like any other procedure or function. However, the global data (DATA.TEXT) is commonly accessible to all routines, including routines which are not part of the unit.

Any modifications to DATA.TEXT must be made to the SOURCE: routines also. If the data structure of the unit is not identical to the data structure of the main program on SOURCE:, then the unit will cause a using program to fail in unpredictable ways.

Note: The PRINT procedure has been left unimplemented in UNIT:PRINT.TEXT. You can implement any print routine desired by modifying the PRINT.TEXT file, recompiling the unit, reinstalling it in SYSTEM.LIBRARY, and then recompiling and linking QUERY.TEXT. This causes the new PRINT procedure to become part of MICROBOOK, so you can output forms, pages, etc. to a printer.

A good model for implementing the PRINT procedure is procedure DISPLAY. This procedure shows how to make sure the book and chapter are open, how to retrieve a form, and how to retrieve and output a page of values. Notice how the variable PR:TEXT has been defined so that it can be used as the printer file. This is shown in UNIT:UNIT.TEXT. Also notice how the printer file is opened in UNIT:UNIT.TEXT as follows:

```
REWRITE(PR,CONCAT(PREFIX,'PRINTER:')) ;
```

which initializes the printer. Thus, the PRINT routine need only use a write intrinsic as follows:

```
WRITELN(PR, (*list of values, etc.*)) ;
```

to output a line. Of course the line must be formatted, properly.

14.2 LISTING

```
(*$S+*)
UNIT MICROBOOK;

INTERFACE
 uses CHAINSTUFF;
 const
   NBOOKS   =002 ;
   NCHAPS   =003 ;                   (* MAX SIZE OF LIBRARY      *)
   NCOLS    =002 ;                   (* MAX SIZE OF BOOK         *)
   NWORDS   =010 ;                   (* NO. ENTRIES/PAGE         *)
   MAXWORD  =015 ;                   (* MAX NO. CHAR/DATA WORD   *)
   NBTREE   =008 ;                   (* ORDER OF B TREE          *)
   NPP      =008 ;                   (* ORDER  OF INDEXES        *)
   MAXKEY   =015 ;
   MAXCHAP  =016 ;
   XORIGIN  =001 ;
   XLIMIT   =041 ;
   YORIGIN  =002 ;
   YLIMIT   =010 ;
   UPPER    =000 ;
   LEFT     =000 ;
   NULL     =  -1 ;
   NONE     =000 ;
   BLANK='                                                                   ';
  type
   XAXIS    = XORIGIN .. XLIMIT ;    YAXIS = YORIGIN .. YLIMIT ;
   FNAMES   = string[ MAXCHAP ] ;   (* FILE NAMES ARE STRINGS *)
   STRKEY   = string[ MAXKEY ]  ;   (* KEY  NAMES ARE STRINGS *)
 STRWORD   = string[ MAXWORD ] ;   (* DATAWORDS ARE STRINGS  *)
 ARYWORD   = array [ 1.. NWORDS ] of STRWORD; (* VALUES OF PAGE *)
 BOOKS     = packed record         (* LIST OF BOOKS IN LIBR. *)
              ITEMS   : 0..NBOOKS ;
              TITLES : array[ 1.. NBOOKS ] of FNAMES ;
              PASSWORD : array[ 1..NBOOKS ] of FNAMES ;
              BOPENED : packed array[1..NBOOKS] of boolean ;
             end ;
CHAPTERS   = packed record          (* CHAPTER HEADINGS      *)
              NCHAP   : 0..NCHAPS ;
              HEADING: array[ 1..NCHAPS ] of FNAMES ;
              PASWORD: array[ 1..NCHAPS ] of FNAMES ;
              COPENED: packed array[ 1..NCHAPS ] of boolean ;
             end ;
WORDS      = packed record          (* format of data dictionary entry *)
              WKEY    : string[ MAXKEY ] ;
              TITLE   : string[ MAXCHAP ] ;
              CHAP    : string[ MAXCHAP ] ;
              COLMN   : 0..NCOLS ;
              XPOS    : XAXIS ;
              YPOS    : YAXIS ;
              LONG    : integer ;
              PLACE   : integer ;
             end ;
PAGES      = packed record           (* FORM AND BLANK SIZES       *)
              NCOL    : 1..NCOLS ;
              MAXPAGE : integer ;   (* max. no. of pages. *)
              PROMPT  : array[ YAXIS ] of string[ XLIMIT ] ;
              NLINES  : YORIGIN..YLIMIT ;
              ENTRY   : array[ 1..NWORDS ] of WORDS ;
             NENTRIES : 1..NWORDS ;
            end ;
DATUM     = packed record            (* entry in ppg file of data   *)
             DSTAT   : boolean ;    (* TRUE=KEEP, FALSE=DELETE     *)
             DPAGENO : integer ;    (* page number *)
             DCOL    : integer ;    (* column number *)
             VALUES  : array[ 1..NWORDS ] of STRWORD ;
             end ;
 NODES    = packed record             (* form of data dictionary nodes *)
             FLAG : boolean ;(* TRUE=NODE CONTAINS DATA POINTERS*)
                             (* FALSE=NODE HAS INDEX POINTERS   *)
             POINTER : array[ 1..NBTREE ] of integer  ;
             SUBKEY  : array[ 1..NBTREE ] of WORDS ;
             LINK    : integer ;
            end ;
 EMPTIES  = packed record      (* header of data dictionary file *)
             TREESIZE : integer ;
             TN       : integer ;     (* NUMBER OF NODES USED    *)
             ROOT     : integer  ;
            end ;
 PPDATA   = string[ MAXWORD ];  (* value stored in index file    *)
 PLOCATION= packed record      (* points to value in data page  *)
             PKEY    : PPDATA ;(* data value                    *)
             PTITLE  : FNAMES ;(* book title        *)
             PHEAD   : FNAMES ;(* chapter heading   *)
             PP      : integer;(* page number       *)
             PCOLUMN : 1..NCOLS ; (* column number *)
            end ;
 PAGEDATA = packed record          (* node in index b-tree files    *)
             PFLAG   : boolean;(* T=leaf, F=internal node       *)
            PPSUBKEY: array[ 1..NPP ] of PLOCATION ;
            PPOINTER: array[ 1..NPP ] of integer ;
            PLINK   : integer ;
           end ;
PUSED    = packed record            (* list of available space of index*)
            PN      : integer;(* number of pages used, so far.  *)
            PROOT   : integer;(* current root of b-tree index   *)
            NEX     : integer;(* next pointer    *)
            NXI     : integer;(* next subscript of key   *)
           end ;

var
  LIBNAME   : string ;          (* name of microbook library       *)
  LIBRARY   : file of BOOKS ;   (* contains the library of books *)
  CHAPHEAD  : FNAMES ;          (* chapter heading name            *)
  CONTENTS  : file of CHAPTERS;(*table of contents of a book    *)
  FORMFILE  : FNAMES ;          (* name of form file for chapter *)
  FORM      : file of PAGES ;   (* file containing form of chapt *)
  BOOK      : FNAMES ;          (* working file name  for books  *)
  FILESPACE : FNAMES ;          (* name of availist file for ... *)
  AVAILIST  : file of EMPTIES;(* list of available space for...*)
  DICTION   : file of NODES ;   (* index file = b-tree structure *)
  PINDEX    : file of PAGEDATA;(* index to pages of a book      *)
  AINDEX    : file of PUSED ;   (* list of available space in index*)
  PPG       : file of DATUM ;   (* data file containing values   *)
  KEYIN     : char ;
  LINEIN    : string ;          (* misc. prompt variables          *)
  SHORT     : string ;          (* working blank line for MAXCHAP*)
  PW        : FNAMES ;          (* password working variable       *)
  NPX       : integer;          (* size of page file (also index)*)
```

```
    PR          : text    ;        (* printer:  text file          *)
    PREWORD     : STRWORD;
    PRECHAP     : FNAMES ;
    PREBOOK     : FNAMES ;
    PREFIX      : FNAMES ;

procedure FPSTR( NUMBER : real; var STRIN : STRWORD );

function STRFP( STRI : STRWORD; var FP : real ) : boolean;

procedure PROMPTS( X, Y : integer; LINE : string; var INPUT : string );

function OPENCHAP( CHAP : FNAMES ) : boolean;

function OPENBOOK( BOOK : FNAMES ) : boolean;

function CHPEN( BOOK, CHAP : FNAMES ) : boolean;

function BKPEN( BOOK : FNAMES ) : boolean;

procedure CPRWORD( WORD : STRWORD; var PREWORD : STRWORD );

procedure CPRCHAP( CHP : FNAMES; var PRECHAP : FNAMES );

procedure DISPLAY( BOOK, CHAP : FNAMES; COL, PAG : integer );

procedure INPUT( BOOK, CHAP : FNAMES );

procedure PRINT( BOOK, CHAP : FNAMES; PAG : integer );
procedure ADD( WORD1, WORD2 : STRWORD; var ADDSTRIN : STRWORD );

procedure SUB( WORD1, WORD2 : STRWORD; var SUBSTRIN : STRWORD );

procedure MUL( WORD1, WORD2 : STRWORD; var MULSTRIN : STRWORD );

procedure DVD( WORD1, WORD2 : STRWORD; var DVDSTRIN : STRWORD );

procedure ZERO( var STRING0 : STRWORD );

procedure FIRST( WORD : STRWORD; var KEY : STRWORD );

function LOOKUP( WORD , KEY : STRWORD ) : integer;

function NEXT( WORD : STRWORD; var KEY : STRWORD ) : integer ;

procedure GETWORD( WORD : STRKEY; var VALUE : STRWORD; PAGE : integer );

procedure PUTWORD( WORD : STRKEY; VALUE : STRWORD;
                   PAGE : integer; XFLAG : boolean  );

procedure GETPAGE( WORD : STRKEY; var VALUE : ARYWORD; PAGE : integer );

procedure PUTPAGE( WORD : STRKEY; VALUE : ARYWORD;
                   PAGE : integer; XFLAG : boolean  );

function DICLOOK( WORD: STRKEY; var BK, CHP : FNAMES; var CLM, LNO : integer )
         : boolean;

IMPLEMENTATION
```

```
var I, J, NODE : integer; LIB : string;

(*$I UNIT: ARITH*)
(*$I UNIT: CPRWORD*)
(*$I UNIT: CPRCHAP*)
(*$I UNIT: DICLOOK*)
(*$I UNIT: PROMPTS*)
(*$I UNIT: OPENCHAP*)
(*$I UNIT: OPENBOOK*)
(*$I UNIT: CHPEN*)
(*$I UNIT: BKPEN*)
(*$I UNIT: PRINT*)
(*$I UNIT: DISPLAY*)
(*$I UNIT: INDEXWORD*)
(*$I UNIT: INPUT*)
(*$I UNIT: GETWORD*)
(*$I UNIT: PUTWORD*)
(*$I UNIT: GETPAGE*)
(*$I UNIT: FIRST*)
(*$I UNIT: PUTPAGE*)
(*$I UNIT: LOOKUP*)
(*$I UNIT: NEXT*)
begin
(*$I UNIT: INIT*)
end.
```

```
(*  UNIT : ARITH  *)

function STRFP;
(*$G+*)
label 10;
const MAXREAL=1.70E37; (* MAX/10 *)
      MINREAL=1.2E-37; (* MIN/10 *)
var DEC,DEX,EDP,INX,LEN: integer;
    FLG,SW,DP,EX,IM,MN,MX,SN: boolean;
    CH: char;
    NUMERIC,EXPONENT,MODIFIER: set of char;

procedure TERMINATE;
  var I: integer;
  begin
    FLG := false;
    if MX then DEX:=-DEX;
    EDP := EDP + DEX - DEC;
    if EDP < 0
       then for I := 1 to -EDP do
            if FP >= MINREAL then FP := FP/10.0
               else FP := 0   (* UNDERFLOW >= 0 *)
       else for I := 1 to EDP do
            if FP <= MAXREAL then FP := FP * 10.0
               else FLG := true;      (* OVERFLOW *)
     if MN then FP := -FP;
     STRFP := true;
     FLG := true   (* SUCCESSFUL CONVERSION *)
   end;
```

```
procedure SEARCH;
(*$G+*)
label 7;
begin
  SW := false;
  while INX <= LEN do
    if STRI[ INX ] IN NUMERIC
       then begin
(*$R-*)
            while ( INX > 1 ) and ( STRI[ INX-1 ] in EXPONENT + MODIFIER)
(*$R+*)
              do INX := INX - 1;
            goto 7                    (* FOUND START OF NUMBER *)
            end
       else INX := INX + 1;
  SW := true;    (* NON-NUMERIC STRING *)
7:
end;

begin   (* STRFP *)
  NUMERIC := [ '0'..'9'];
  EXPONENT := [ 'E','e' ];
  MODIFIER := [ '+','-','.',',' ];
  DP := false; EX := false; IM := true;
  MN := false; MX := false; SN := false;
  DEC := 0; DEX := 0; EDP := 0; INX := 1;
  LEN := LENGTH (STRI); FP := 0;
  STRFP := false;
  SEARCH;
  if SW then  goto 10;  (* find start of number *)
```

```
while INX <= LEN do begin
  CH := STRI [ INX ];
  if CH in NUMERIC + EXPONENT + MODIFIER
     then begin
       if CH in NUMERIC
          then if  EX
            then begin
              if DEX < 1000 then
                 DEX := DEX * 10 + ord ( CH ) - ord ( '0' );
              SN := TRUE
            end
            else begin
              if FP < 1.0E8
                 then FP := FP * 10 + ord ( CH ) - ord ( '0' )
                 else EDP := EDP + 1;
              if DP then DEC := DEC + 1;
              IM := false;
              SN := true
            end
          else case CH of
              '+' : if SN then
                      begin
                        TERMINATE;
                        if FLG then
                          goto 10
                      end
                    else SN := true;
              '-' : if SN then
                      begin
                        TERMINATE;
```

```
             if FLG then
               goto 10
           end
         else begin
                if EX then MX := true
                else MN := true;
              end;
    '.' : if DP or EX then
            begin
              TERMINATE;
              if FLG then
                goto 10
            end
          else DP := TRUE;
    'E','e' : if EX then
                begin
                  TERMINATE;
                  if FLG then
                    goto 10
                end
              else begin
                  if IM then FP := 1.0;
                  EX := true;
                  SN := false
             end;
     end; (* case *)
INX := INX + 1
end
else begin
       TERMINATE;
```

```
                    if FLG then goto 10    (* END OF NUMBER *)
                  end   (* if CH *)
         end;      (* while *)
   TERMINATE;      (* END OF STRING *)
10:
end;

procedure FPSTR;
var
  NOFRAC : real; NOFRAC2, NOINT : integer;
  NSIGN : boolean; STR1, STR2 : string[ MAXWORD ];
begin
    if NUMBER >= 0.0 then
      NSIGN := false
    else begin
           NUMBER := -1.0 * NUMBER;
           NSIGN  := true
         end;
    NOINT := trunc( NUMBER );
    STR( NOINT, STR1 );
    NOFRAC := NUMBER - NOINT;
    if NOFRAC < 1.00000E-3 then
      begin
        NOFRAC2 := trunc( NOFRAC * 10000.0);
        STR( NOFRAC2, STR2);
        STRIN := CONCAT( STR1, '.000', STR2)
      end
    else if NOFRAC < 1.00000E-2 then
           begin
             NOFRAC2 := trunc( NOFRAC * 10000.0);
             STR( NOFRAC2, STR2);
             STRIN := CONCAT( STR1, '.00', STR2)
           end
         else if NOFRAC < 1.00000E-1 then
                begin
                  NOFRAC2 := trunc( NOFRAC * 10000.0);
                  STR( NOFRAC2, STR2);
                  STRIN := CONCAT( STR1, '.0', STR2)
                end
              else begin
                     NOFRAC2 := trunc( NOFRAC * 10000.0);
                     STR( NOFRAC2, STR2 );
                     STRIN := CONCAT( STR1, '.', STR2 )
                   end;
    if NSIGN = true then STRIN := CONCAT( '-',STRIN)
end;

procedure ADD;
var ADDFP, FP1, FP2 : real;
begin
  if STRFP( WORD1, FP1 ) and STRFP( WORD2, FP2 ) then
    begin
      ADDFP := FP1 + FP2;
      FPSTR( ADDFP, ADDSTRIN )
    end
  else begin
        writeln('Illegal parameters of ADD. Strike <rtn>.');
        read( KEYIN )
       end
end;
```

```
procedure SUB;
var SUBFP, FP1, FP2 : real;
begin
  if STRFP( WORD1, FP1 ) and STRFP( WORD2, FP2 ) then
    begin
      SUBFP := FP1 - FP2;
      FPSTR( SUBFP, SUBSTRIN )
    end
  else begin
        writeln(’Illegal parameters of SUB. Strike <rtn>.’);
        read( KEYIN )
       end
end;

procedure MUL;
var MULFP, FP1, FP2 : real;
begin
  if STRFP( WORD1, FP1 ) and STRFP( WORD2, FP2 ) then
    begin
      MULFP := FP1 * FP2;
      FPSTR( MULFP, MULSTRIN )
    end
  else begin
        writeln(’Illegal parameters of MUL. Strike <rtn>.’);
        read( KEYIN )
       end
end;

procedure DVD;
var DVDFP, FP1, FP2 : real;
```

```
begin
  if STRFP( WORD1, FP1 ) and STRFP( WORD2, FP2 ) then
    begin
      DVDFP := FP1 / FP2;
      FPSTR( DVDFP, DVDSTRIN )
    end
  else begin
        writeln('Illegal parameters of DVD. Strike <rtn>.');
        read( KEYIN )
       end
end;

procedure ZERO;
  begin
    STRING0 := '0'
  end;

(*    END OF ARITH  *)

procedure CPRWORD;
begin
  if PREWORD <> WORD then
    begin
      if PREWORD <> ' ' then
        begin SEEK( AINDEX, 0 ); PUT( AINDEX ); end ;
      CLOSE( PINDEX, LOCK ); CLOSE( AINDEX, LOCK );
      RESET( PINDEX, CONCAT( PREFIX, WORD, '.INDEX' ) );
      RESET( AINDEX, CONCAT( PREFIX, WORD, '.AVAIL' ) );
      PREWORD := WORD
    end;
end;
```

```
procedure CPRCHAP;
begin
  if PRECHAP <> CHP then
    begin
      CLOSE( FORM, LOCK ); CLOSE( PPG, LOCK );
      RESET( FORM, CONCAT( PREFIX, CHP, '.FORM' ) );
      RESET( PPG, CONCAT( PREFIX, CHP, '.PAGES' ) );
      PRECHAP := CHP
    end;
end;


function DICLOOK;
(*$G+*)
label 7;
var I, NODE : integer;
begin
  SEEK( AVAILIST, 0 ); GET( AVAILIST );
  NODE := AVAILIST^.ROOT;
  repeat
    I := 1;
    SEEK( DICTION, NODE );
    GET( DICTION );
    with DICTION^ do
      begin
        while ( SUBKEY[ I ].WKEY < WORD ) and
              ( SUBKEY[ I ].WKEY <> COPY( BLANK, 1, MAXKEY ) ) do
          I := SUCC( I );
        if SUBKEY[ I ].WKEY = WORD then
          begin
```

```
            BK   := SUBKEY[ I ].TITLE;
            CHP := SUBKEY[ I ].CHAP;
            CLM := SUBKEY[ I ].COLMN;
            LNO := SUBKEY[ I ].PLACE;
            DICLOOK := true;
            goto 7
          end
        else
          NODE := POINTER[ I ]
      end  (* with *)
  until ( NODE = NULL ) or ( DICTION^.FLAG );
  DICLOOK := false;
  7:
end;



 procedure PROMPTS;
 var
   KEYIN   : char ;
   I       : integer ;

   begin
    repeat
     GOTOXY( X,Y ) ; WRITE( LINE ) ; READLN( INPUT ) ;
     GOTOXY( X,Y ) ; WRITE( BLANK ) ; GOTOXY( X,Y ) ;
     WRITE( 'Are you sure ?' ) ; READLN( KEYIN ) ;
    until KEYIN = 'Y' ;
   end ;  (* PROMPTS *)
```

```
function OPENCHAP;
 var
  ITEM, IT : integer ;
  FOUND : boolean ;

 begin
   PAGE( OUTPUT );
   OPENCHAP := true ;
   with CONTENTS^  do
    begin
     FOUND := FALSE ;
     for ITEM := 1 to NCHAP do
      if HEADING[ ITEM ] = CHAP
       then begin
              FOUND := TRUE; IT := ITEM;
              COPENED[ ITEM ] := true;
            end ;  (* for loop *)
     repeat
       GOTOXY( LEFT, UPPER ); WRITE( 'Enter ', CHAP,'  password : ' );
       READLN( KEYBOARD, LINEIN ); GOTOXY( LEFT, UPPER ); WRITE( BLANK );
       GOTOXY( LEFT, UPPER ); WRITE( ' Are you sure ? ' ); READLN( KEYIN );
     until KEYIN = 'Y';
     PW := LINEIN ;
     if ( PASWORD[ IT ] <> PW ) or ( not FOUND )
         then
          begin
           OPENCHAP := FALSE ;
           GOTOXY( LEFT, UPPER); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
           if FOUND
            then WRITE( 'Wrong chapter password. Strike <return>' )
            else WRITE( 'Chapter not found. Strike <return>.' ) ;
           READ( KEYIN ) ;
          end ;  (* if  *)
    end; (* with *)
  PAGE( OUTPUT )
end;  (* OPENONE *)


function OPENBOOK;
var
  ITEM, IT : integer ;
  FOUND : boolean ;

begin
  PAGE( OUTPUT ); OPENBOOK := true;
  with LIBRARY^ do
   begin
    FOUND := FALSE ;
    for ITEM := 1 to ITEMS  do
     if TITLES[ ITEM ] = BOOK
      then begin
             FOUND := TRUE ; IT := ITEM;
             BOPENED[ ITEM ] := true;
           end; (* for loop *)
    if FOUND
     then
      begin
       repeat
         GOTOXY( LEFT, UPPER ); WRITE( 'Enter ', BOOK, ' password : ' );
         READLN( KEYBOARD, LINEIN ); GOTOXY( LEFT, UPPER ); WRITE( BLANK );
         GOTOXY( LEFT, UPPER ); WRITE( ' Are you sure ? ' ); READLN( KEYIN );
       until KEYIN = 'Y';
       if PASSWORD[ IT ] <> LINEIN
        then
         begin
          OPENBOOK := false;
          GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
          WRITE( 'Wrong book password. Strike <return>. ' ) ; READ( KEYIN ) ;
         end ; (* if *)
      end  (* then *)
     else
      begin
       OPENBOOK := false;
       GOTOXY( LEFT, UPPER) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
       WRITE( 'Book not found. Strike <return>. ' ) ; READ( KEYIN ) ;
      end ; (* else *)
   end;     (* end *)
   (*  BOOK AND PASSWORD O.K. *)
 PAGE( OUTPUT )
end ;  (*   OPENBOOK   *)


function CHPEN;
(*$G+*)
label 7;
var I : integer;
begin
  if PREBOOK <> BOOK then
    begin
      CLOSE( CONTENTS, LOCK );
      RESET( CONTENTS, CONCAT( PREFIX, BOOK, '.BOOK' ) );
      PREBOOK := BOOK
    end;
  with CONTENTS^ do
  for I := 1 to NCHAP do
    if HEADING[ I ] = CHAP then
      if COPENED[ I ] then begin CHPEN := true; goto 7 end
      else if not OPENCHAP( CHAP ) then begin CHPEN := false; goto 7 end
           else begin
                  if COPENED[ I ] then
                    begin CHPEN := true; goto 7 end
                  else begin CHPEN := false; goto 7 end
                end;
  writeln(' Chapter : ', CHAP, ' not found ' );
  write( ' Strike <rtn> to exit.. ' );
  read( KEYIN ); CHPEN := false;
7:
end;


function BKPEN;
(*$G+*)
label 7;
var I : integer;
begin
  with LIBRARY^ do
    for I := 1 to ITEMS do
      if TITLES[ I ] = BOOK then
        if BOPENED[ I ] then begin BKPEN := true; goto 7 end
        else if not OPENBOOK( BOOK ) then begin BKPEN := false; goto 7 end
```

```
             else begin
                    if BOPENED[ I ] then
                      begin BKPEN := true; goto 7 end
                    else begin BKPEN := false; goto 7 end
                  end;
  writeln(' BOOK : ', BOOK, ' not found' );
  write(' Strike <rtn> to exit..' );
  read( KEYIN ); BKPEN := false;
  7:
end;


procedure PRINT;
Begin
 (*  due to the unique nature of printers...                       *)
 WRITELN( 'PRINT not implemented. See UNIT:PRINT.TEXT ' ) ;
 WRITE( 'Strike <rtn> ' ) ;
 READ( KEYIN ) ;
 (*  This must be included in a recompiler unit                    *)
end;


procedure DISPLAY;
(*$G+*)
label 7;
var I, M : integer;
begin
  if not BKPEN( BOOK ) then goto 7;
  if not CHPEN( BOOK, CHAP ) then goto 7;
  CPRCHAP( CHAP, PRECHAP );
  SEEK( FORM, COL - 1 );GET( FORM );
  if COL > FORM^.NCOL then
    begin
      writeln(' Column : ', COL, ' does not exist.');
      write(' Strike <rtn> to exit.. ' ); read( KEYIN ); goto 7
    end;
 SEEK( PPG, PAG * FORM^.NCOL + COL - 1 ); GET( PPG );
 if PAG <> PPG^.DPAGENO then
   begin
     writeln( ' PAGE : ', PAG, ' does not exist.');
     write( ' Strike <rtn> to exit.. ' ); read( KEYIN ); goto 7
   end;
  PAGE( OUTPUT );GOTOXY( LEFT, YORIGIN );
  with FORM^ do
    begin
      for I := YORIGIN to NLINES do
          writeln( PROMPT[ I ] );
      GOTOXY( LEFT, UPPER ); write( BLANK ); GOTOXY( LEFT, UPPER );
      for I := 1 TO NENTRIES do
        begin
          GOTOXY( ENTRY[ I ].XPOS, ENTRY[ I ].YPOS );
          writeln( PPG^.VALUES[ I ] )
        end   (* for *)
    end;    (* with *)
7:
end;
```

```
(*  UNIT : INDEXWORD  *)

procedure INDEXWORD( DWORD : PLOCATION ; BTREE : FNAMES );
  (*$G+ *)
    label
      4, 10 ;

    const
     NULL = -1 ;

    var
      I, K, LAST : integer ;      (* WORKING COUNTERS, ETC            *)
      NEW, TEMP  : PAGEDATA  ;       (* WORKING NODES                   *)
      NEWNODE,
      NODE,
      TEMPPTR    : integer  ;     (* WORKING POINTERS                 *)
      STOP       : -1 .. 20 ;     (* STACK PPOINTER                    *)
      STACK      : array[ 0..20 ] of integer ;
      KEY        : PLOCATION;     (* THE SEARCH KEY                   *)
      ITEM       : integer  ;     (* THE POINTER TO BTREE NODES FILE*)


 function ALOCATE : integer ;
   begin
     with AINDEX^ do
      begin
        PN := PN + 1 ;
        ALOCATE := PN;
      end  (*  with  *)
 end;                        (*  ALOCATE    *)
```

```
procedure FINISH ;
 Begin
  SEEK( AINDEX, 0 ) ;
  PUT ( AINDEX ) ;
 end ;   (* FINISH  *)

function  LOOKUP : boolean ;
  begin
    LOOKUP := TRUE ;
    STOP := 0 ; STACK[ STOP ] := NULL ;
    STOP := 1 ; STACK[ STOP ] := NULL ;
    NODE := AINDEX^.PROOT ;        (* START AT PROOT NODE        *)
    repeat
      I := 1 ;
      SEEK( PINDEX, NODE ) ;
      GET ( PINDEX ) ;
      with PINDEX^ do
        begin
          while ( PPSUBKEY[ I ].PKEY <> COPY( BLANK, 1, MAXWORD )  )
                      and
                ( PPSUBKEY[ I ].PKEY < KEY.PKEY )
           do
            begin
            I := SUCC( I );   (* FIND INSERT PLACE            *)
            end ;
          STOP := STOP + 1 ; STACK[ STOP ] := NODE ;
          STOP := STOP + 1 ; STACK[ STOP ] := I ;
          NODE := PPOINTER[ I ] ;
          if PPSUBKEY[ I ].PKEY = KEY.PKEY
             then
```

```
              begin
               LOOKUP := FALSE ;
              end ;
        end ;                           (* with *)
    until ( NODE = NULL ) or ( PINDEX^.PFLAG );
  end ;                                 (*  LOOKUP   *)

procedure SPLIT ;
  begin
    with  PINDEX^ do
      begin
        NEW.PFLAG := PFLAG ; NEW.PLINK := PLINK;
        LAST     := (NPP + 1) div 2 ;
        K        := 1 ;
        while LAST+K <= NPP do
          begin
            NEW.PPOINTER[ K ] := PPOINTER[ LAST + K ] ;
            NEW.PPSUBKEY [ K ] := PPSUBKEY [ LAST + K ] ;
            K := SUCC ( K ) ;
          end ;              (* while *)
        NEW.PPOINTER[ K ] := TEMPPTR ;
        NEW.PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
        for K := K+1 to NPP do
          begin
            NEW.PPOINTER[ K ] := NULL ;
            NEW.PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
          end ;              (* while *)
      end ;                (* with  *)

   (*   STEP   7   *)
```

```
     TEMP    := PINDEX^ ;
     PINDEX^  := NEW ;
     NEWNODE := ALOCATE ;
     SEEK( PINDEX, NEWNODE ) ;
     PUT ( PINDEX ) ;
     PINDEX^  := TEMP ;
     with PINDEX^ do
       begin
         KEY.PKEY  := PPSUBKEY[ LAST ].PKEY ;
         for K := LAST+1  to NPP do
           begin
             PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
             PPOINTER[ K ] := NULL ;
           end ;                      (*  for  *)
         PLINK := NEWNODE ;
       end ;              (* with *)
     SEEK( PINDEX, NODE ) ;
     PUT ( PINDEX ) ;
end ;               (* SPLIT *)

  procedure NEWROOT ;
    begin
      with  PINDEX^ do
        begin
          PFLAG := FALSE ;
          PPOINTER[ 1 ] := ITEM ;
          PPSUBKEY [ 1 ] := KEY ;
          PPOINTER[ 2 ] := NEWNODE ;
          PPSUBKEY [ 2 ].PKEY := COPY( BLANK, 1, MAXWORD );
          for K := 3 to NPP do
```

```
            begin
              PPOINTER[ K ] := NULL ;
              PPSUBKEY [ K ].PKEY := COPY( BLANK, 1, MAXWORD ) ;
            end ;
          PLINK := NULL ;
          NEWNODE := ALOCATE ;
        end ;          (* with *)
        SEEK( PINDEX, NEWNODE ) ;
        PUT ( PINDEX ) ;
        AINDEX^.PROOT := NEWNODE ;
   end  (*  NEWROOT  *) ;

    (******    BODY OF INDEXWORD     *****)

begin
 CPRWORD( BTREE, PREWORD );
 SEEK( PINDEX, 0 ); GET( PINDEX );
 GOTOXY( LEFT, UPPER ); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
 WRITE( 'Indexing : ', DWORD.PKEY, ' in file : ', BTREE, '.INDEX' ) ;
 KEY := DWORD ;
 ITEM := NULL ;
 if not LOOKUP then WRITE(' Duplicate' ) ;
 I        := STACK[ STOP ] ; STOP := STOP - 1 ;
 NODE     := STACK[ STOP ] ; STOP := STOP - 1 ;
 NEW.PFLAG := TRUE ;

  (*     STEP   4      *)

  4:
   with PINDEX^ do
```

```
begin
  TEMPPTR := PPOINTER[ NPP ] ;
  for K := NPP downto I + 1 do
    begin
      PPOINTER[ K ] := PPOINTER[ K - 1 ] ;
      PPSUBKEY [ K ] := PPSUBKEY [ K - 1 ] ;
    end ;
  PPOINTER[ I ] := ITEM ;
  PPSUBKEY [ I ] := KEY ;
  PFLAG          := NEW.PFLAG ;
end ;             (*  with  *)

(*   STEP   5    *)

if PINDEX^.PPSUBKEY[ NPP ].PKEY = COPY( BLANK, 1, MAXWORD )
  then
    begin
      SEEK( PINDEX, NODE ) ;
      PUT ( PINDEX ) ;
      FINISH ;
      goto 10;
    end ;

SPLIT ;

ITEM   := NODE ;
I      := STACK[ STOP ] ; STOP := STOP - 1 ;
NODE   := STACK[ STOP ] ; STOP := STOP - 1 ;
if NODE = NULL
  then
```

```
         begin
         NEWROOT ;
         FINISH ;
         goto 10;
       end ;     (* IF-THEN  *)
  SEEK( PINDEX, NODE ) ;
  GET ( PINDEX ) ;
  PINDEX^.PPOINTER[ I ] := NEWNODE ;
  NEW.PFLAG := FALSE ;
  goto 4 ;
10:
end;                    (* INDEXWORD *)


(*   UNIT : INPUT   *)

procedure INPUT;
(*$G+*)
label 11;
var
  COL, ITEM, IT : integer ;
  FOUND     : boolean ;
  DATAWORD : string[ MAXWORD ] ;
  OPENFILE : string[ MAXCHAP ] ;

procedure FILLFORM;

procedure GETFORM( COL : integer ) ;
 begin
  SEEK( FORM, COL - 1 );
  GET( FORM ) ; GOTOXY( LEFT, YORIGIN ) ;
  with  FORM^ do
   begin
    for ITEM := YORIGIN to NLINES do
     begin
      if POS( '/', PROMPT[ ITEM ] ) = 0
       then WRITELN( PROMPT[ ITEM ] );
    end ; (* for  *)
   GOTOXY( LEFT, UPPER ); WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
   WRITE( 'Strike <return> when ready to enter column = ', COL ) ;
   READ( KEYIN ) ;
  end; (*  with  *)
end ;   (* GETFORM  *)

procedure GETWORDS( COL : integer ) ;
 var
  I : integer ;
  LINE : string ;
  NOLINE : boolean ;
 begin
  ITEM := 0 ;
  for I := YORIGIN to FORM^.NLINES do
   begin
    LINE :=CONCAT( FORM^.PROMPT[ I ], '.' ) ;
    NOLINE := POS( '/', LINE ) = 1 ;
    repeat
     if POS( '-', LINE ) > 0
      then
       begin
        DELETE( LINE, 1, POS( '-', LINE ) ) ;
        ITEM := ITEM + 1 ;
        while POS( '-', LINE ) = 1 do DELETE( LINE, 1, 1 ) ;
        if not NOLINE
         then
          begin
           with FORM^.ENTRY[ ITEM ] do
            begin
             GOTOXY( XPOS, YPOS ) ; READLN( LINEIN ) ;
             if LENGTH( LINEIN ) > LONG
              then DATAWORD := COPY( LINEIN, 1, LONG )
              else DATAWORD := LINEIN ;
            with PPG^ do
             begin
              VALUES[ ITEM ] := DATAWORD;
              DPAGENO := IT ; DCOL := COL ;
             end ;  (*  with  *)
            end ;  (*  with  *)
           end ;  (* if not NOLINE  *)
         end ; (*  if POS('-',LINE) then  *)
    until POS( '-', LINE ) = 0 ;
   end ;  (*  for I  *)
 end ;  (* GETWORDS  *)

procedure NEWORD ( I : integer ) ;
 begin
  with FORM^.ENTRY[ I ] do
   begin
    GOTOXY( XPOS, YPOS ) ; WRITE( COPY( BLANK, 1, LONG ) ) ;
    GOTOXY( XPOS, YPOS ) ; READLN( LINEIN ) ;
    if LENGTH( LINEIN ) > LONG
    then DATAWORD := COPY( LINEIN, 1, LONG )
    else DATAWORD := LINEIN ;
   PPG^.VALUES[ I ] := DATAWORD ;
  end ;   (*  with  *)
end ;   (*   NEWORDS  *)

procedure PUTDATA ;
 var
  BTREE  : FNAMES ;
  DATAWORD : PLOCATION ;

 begin
  with FORM^ do
   for ITEM := 1 to NENTRIES do
    begin
     GOTOXY( ENTRY[ ITEM ].XPOS, ENTRY[ ITEM ].YPOS ) ;
     WRITE( COPY( BLANK, 1, ENTRY[ ITEM ].LONG ) ) ;
    if POS( '*', ENTRY[ ITEM ].WKEY ) = 1
     then
      begin
       BTREE := COPY( ENTRY[ ITEM ].WKEY, 2, LENGTH( ENTRY[ ITEM ].WKEY)-1);
       with DATAWORD do
        begin
         PKEY := PPG^.VALUES[ ITEM ] ;
         PTITLE := BOOK ;
         PHEAD  := CHAPHEAD ;
         PP   := IT ;
         PCOLUMN := COL ;
         INDEXWORD( DATAWORD, BTREE ) ;
        end ;
       end ;  (* if  *)
     end ;  (*  for  *)
    SEEK( PPG, IT * FORM^.NCOL + COL - 1 ) ;
    PUT( PPG ) ;       (* STORE DATA WORD IN PAGE FILE  *)
    GOTOXY( LEFT, UPPER ); WRITE( BLANK ); GOTOXY( LEFT, UPPER );
    WRITE( 'Page ', IT, ' entered. Strike <return>.' ) ;
    READ( KEYIN ) ;
  end ;  (* putdata  *)

begin      (* F I L L F O R M    B O D Y   *)
    PAGE( OUTPUT ) ;
    OPENFILE := COPY( BLANK, 1, MAXCHAP ) ;
    GOTOXY( LEFT, UPPER); write( BLANK ); GOTOXY( LEFT, UPPER );
    write( 'Strike <rtn> when ready to enter page = ', IT ); read( KEYIN );
    repeat    (* enter as many pages as you want  *)
     for COL := 1 to FORM^.NCOL do
      begin
       PAGE( OUTPUT ) ;
       GETFORM( COL ) ;    (* DISPLAY FORM ON SCREEN *)
       GETWORDS( COL );    (* GET DATA VALUES FROM USER *)
       repeat
        GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER ) ;
        WRITE( 'Enter number to change, or zero to stop : ' ) ;
        READLN( ITEM ) ;
        if ( ITEM <> 0 ) and ( ITEM <= FORM^.NENTRIES ) then NEWORD( ITEM ) ;
       until ITEM = 0 ;
       PUTDATA ;          (* WRITE DATA TO DISK   *)
      end ;  (* for COL *)
      IT := IT + 1 ;       (* NEXT PAGE  *)
      PROMPTS( LEFT, UPPER, ' Enter S(top, A(dd more pages : ', LINEIN );
   if COPY( LINEIN, 1, 1 ) = 'S' then
    KEYIN := 'S'
   else
    if IT > FORM^.MAXPAGE then
     begin
      write( '*** Chapter OVERFLOW ***, Strike <rtn>.. ' );
      read( KEYIN ); KEYIN := 'S'
     end;
  until KEYIN = 'S' ;
  PAGE( OUTPUT );
 end ;  (*   FILLFORM  *)

begin
 if not CHPEN( BOOK, CHAP ) then goto 11;
 CPRCHAP( CHAP, PRECHAP );
 SEEK( PPG, 0 ); GET( PPG );
 while ( not EOF( PPG ) ) and ( PPG^.DPAGENO <> NULL ) do
  begin
   IT := PPG^.DPAGENO;
   GET( PPG )
  end;
 IT := IT + 1;
 SEEK( FORM, 0 ); GET( FORM );
 if IT > FORM^.MAXPAGE then
  begin
   write(' Chapter OVERFLOW, Strike <rtn>.. ');
   read( KEYIN ); CLOSE( FORM ); CLOSE( PPG );
   goto 11
  end;
 FILLFORM;
```

```
11:
end;




procedure GETWORD;
var I, LNO, CLM, NODE : integer; BOK , CHP : FNAMES;
begin  (* GETWORD *)
  if DICLOOK( WORD, BOK, CHP, CLM, LNO ) then
    if BKPEN( BOK ) then
      if CHPEN( BOK, CHP ) then
        begin
          CPRCHAP( CHP, PRECHAP );
          SEEK( FORM, CLM - 1 );
          GET( FORM );
          SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 );
          GET( PPG );
          if not PPG^.DSTAT then
            begin
              writeln(' Data of : ', WORD, ' at page : ', PAGE );
              writeln(' of chapter : ', CHP, ' was deleted ' );
              write(' Type <rtn> to exit.. ' );
              read( KEYIN ); ZERO( VALUE );
            end
         else VALUE := PPG^.VALUES[ LNO ];
      end
end;
```

```
(*   UNIT : PUTWORD  *)

procedure PUTWORD;
var I, LNO, CLM, NODE : integer; BOK, CHP : FNAMES; DATAWORD : PLOCATION ;
begin  (* PUTWORD *)
 if DICLOOK( WORD, BOK, CHP, CLM, LNO ) then
  if BKPEN( BOK ) then
   if CHPEN( BOK, CHP ) then
    begin
     CPRCHAP( CHP, PRECHAP );
     SEEK( FORM, CLM - 1 ); GET( FORM );
     SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 ); GET( PPG );
     if not PPG^.DSTAT then
      begin
       writeln(' Data of : ', WORD, ' at page : ', PAGE );
       writeln(' of chapter : ', CHP, ' was deleted ');
       write(' Strike <rtn> to exit.. ' );
       read( KEYIN );
      end
    else
      begin
       PPG^.VALUES[ LNO ] := VALUE;
       PPG^.DPAGENO := PAGE;
       SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 );
       PUT( PPG );
       if XFLAG and  ( POS( '*', WORD ) > 0 )
        then
         begin
          with DATAWORD do
           begin
```

```
            PKEY    := VALUE ;
            PTITLE  := BOK ;
            PHEAD   := CHP ;
            PP      := PAGE ;
            PCOLUMN := CLM ;
           end ;
          INDEXWORD( DATAWORD, COPY( WORD, 2, LENGTH( WORD ) - 1 ) ) ;
        end ;
       end
    end
end;


procedure GETPAGE;
var I, LNO, CLM, NODE : integer; BOK, CHP : FNAMES;
begin  (* GETPAGE *)
  if DICLOOK( WORD, BOK, CHP, CLM, LNO ) then
   if BKPEN ( BOK ) then
    if CHPEN( BOK, CHP ) then
     begin
      CPRCHAP( CHP, PRECHAP ); SEEK( FORM, CLM - 1 ); GET( FORM );
      SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 ); GET( PPG );
      if not PPG^.DSTAT then
       begin
        writeln(' Data of : ', WORD, ' at page : ', PAGE );
        writeln(' of chapter : ', CHP, ' was deleted' );
        write( 'Strike <rtn> to exit.. ' ); read( KEYIN );
        for I := 1 to FORM^.NENTRIES do ZERO( VALUE[ I ] ) ;
       end
      else
```

```
        begin
         for I := 1 to FORM^.NENTRIES do
          VALUE[ I ] := PPG^.VALUES[ I ];
        end
    end
end;


(*   UNIT :  PUTPAGE  *)

procedure PUTPAGE;
var I, LNO, CLM, NODE : integer; BOK, CHP : FNAMES; DATAWORD : PLOCATION ;
begin  (* PUTPAGE *)
 if DICLOOK( WORD, BOK, CHP, CLM, LNO ) then
  if BKPEN( BOK ) then
   if CHPEN( BOK, CHP ) then
    begin
     CPRCHAP( CHP, PRECHAP );
     SEEK( FORM, CLM - 1 ); GET( FORM );
     SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 );
     GET( PPG );
     if not PPG^.DSTAT then
      begin
       writeln(' Data of : ', WORD, ' at page : ', PAGE );
       writeln(' of chapter : ', CHP, ' was deleted ' );
       write(' Strike <rtn> to exit.. ' );
       read( KEYIN );
      end
     else
      begin
```

```
       for I := 1 to FORM^.NENTRIES do
        begin
         PPG^.VALUES[ I ] := VALUE[ I ];
         if XFLAG and ( POS( '*', FORM^.ENTRY[ I ].WKEY ) > 0 )
          then
           begin
            with DATAWORD do
             begin
              PKEY    := VALUE[ I ] ;
              PTITLE := BOK ;
              PHEAD  := CHP ;
              PP     := PAGE;
              PCOLUMN:= CLM ;
             end ;
            INDEXWORD( DATAWORD, COPY( FORM^.ENTRY[ I ].WKEY, 2,
                       LENGTH( FORM^.ENTRY[ I ].WKEY ) - 1 ) ) ;
           end ;
        end ;    (*  for i   *)
       PPG^.DPAGENO := PAGE;
       SEEK( PPG, PAGE * FORM^.NCOL + CLM - 1 );
       PUT( PPG );
      end
   end
end;


procedure FIRST;
var NODE, CLM, LNO : integer; BK, CHP : FNAMES;

begin (* FIRST *)
```

```
if not DICLOOK( WORD, BK, CHP, CLM, LNO ) then
  begin
    ZERO( KEY );
  end
 else begin
   DELETE( WORD, 1, 1 );
   CPRWORD( WORD, PREWORD );
   NODE := AINDEX^.PROOT;
  repeat
    SEEK( PINDEX, NODE);
    GET( PINDEX );
    AINDEX^.NEX := NODE ;
    with PINDEX^ do
      NODE := PPOINTER[ 1 ];
  until ( NODE = NULL ) and ( PINDEX^.PFLAG ) ;
  AINDEX^.NXI := 2 ;
  KEY := PINDEX^.PPSUBKEY[ 1 ].PKEY;
 end  (* if-then-else  *)
end; (* FIRST *)



function LOOKUP;
(*$G+*)
label 7;
var I, NODE : integer;
begin
  DELETE( WORD, 1, 1 );
  CPRWORD( WORD, PREWORD );
  I := 1;
  NODE := AINDEX^.PROOT ;
  repeat
    SEEK( PINDEX, NODE );
    GET( PINDEX );
    with PINDEX^ do
      begin
        while ( PPSUBKEY[ I ].PKEY < KEY )
          and ( PPSUBKEY[ I ].PKEY <> COPY( BLANK, 1, MAXWORD ) ) do
          I := SUCC( I );
        if PPSUBKEY[ I].PKEY = KEY then
          begin
            LOOKUP := PPSUBKEY[ I ].PP;
            AINDEX^.NEX := NODE;
            AINDEX^.NXI := I + 1;
            goto 7
          end;
        if PLINK <> NULL then
          NODE := PLINK
        else begin
               NODE := PPOINTER[ I ]; I := 1
             end
      end (* with *)
  until  ( NODE = NULL ) or ( PINDEX^.PFLAG );
  LOOKUP := NULL;
  7:
end; (* LOOKUP *)
```

```
function  NEXT;
(*$G+*)
label 7;
var I : integer;

begin
  DELETE( WORD, 1, 1 );
  CPRWORD( WORD, PREWORD );
  I := AINDEX^.NXI;
  SEEK( PINDEX, AINDEX^.NEX );
  GET( PINDEX );
  with PINDEX^ do
   begin
    if I < NPP then
      if PPSUBKEY[ I ].PKEY <> COPY( BLANK, 1, MAXWORD ) then
        begin
          KEY := PPSUBKEY[ I ].PKEY;
          NEXT := PPSUBKEY[ I ].PP ;
          AINDEX^.NXI := I+1 ;
          goto 7
        end;
    if PLINK <> NULL then
      AINDEX^.NEX := PLINK
    else
      AINDEX^.NEX := PPOINTER[ I ];
    if AINDEX^.NEX <> NULL then
      begin
              I := 1;
              SEEK( PINDEX, AINDEX^.NEX );
              GET( PINDEX );
              KEY := PPSUBKEY[ I ].PKEY;
              NEXT := PPSUBKEY[ I ].PP ;
              AINDEX^.NXI := I+1;
       end
     else
       begin
              ZERO( KEY );
              NEXT := NULL ;
       end (* if *)
     end; (* with *)
  7:
end;


CLOSE( AINDEX, LOCK ); CLOSE( PINDEX, LOCK );
CLOSE( CONTENTS, LOCK ); CLOSE( FORM, LOCK );
CLOSE( PPG, LOCK ); CLOSE( LIBRARY, LOCK );
CLOSE( DICTION, LOCK ); CLOSE( AVAILIST, LOCK );
CLOSE( PR ); GETCVAL( LIBNAME );
LIB := LIBNAME; I := 1;
while COPY( LIB, 1, 1 ) <> ':' do
  begin
    DELETE( LIB, 1, 1 ); I := I + 1
  end;
PREFIX := COPY( LIBNAME, 1, I );
RESET( LIBRARY, CONCAT( LIBNAME, '.LIB' ) );
with LIBRARY^ do
  for I := 1 to ITEMS do
    BOPENED[ I ] := false;
```

```
for I := 1 to LIBRARY^.ITEMS do
 begin
   RESET( CONTENTS, CONCAT( PREFIX, LIBRARY^.TITLES[ I ], '.BOOK' ) );
   with CONTENTS^ do
     for J := 1 to NCHAP do
       COPENED[ J ] := false;
   SEEK( CONTENTS, 0 ); PUT( CONTENTS ); CLOSE( CONTENTS ) ;
 end;
RESET( DICTION, CONCAT( LIBNAME, '.DICT' ) );
RESET( AVAILIST, CONCAT( LIBNAME, '.AVAIL' ) );
NODE := AVAILIST^.ROOT; I := 1;
repeat
  SEEK( DICTION, NODE ); GET( DICTION );
  with DICTION^ do
  begin
  while ( SUBKEY[ I ].WKEY <> COPY( BLANK, 1, MAXKEY ) )
        and ( COPY( SUBKEY[ I ].WKEY, 1, 1 ) <> '*' ) do
          I := SUCC( I );
  if COPY( SUBKEY[ I ].WKEY, 1, 1 ) = '*' then
    begin
      DELETE( SUBKEY[ I ].WKEY, 1, 1 );
      RESET( AINDEX, CONCAT( PREFIX, SUBKEY[ I ].WKEY, '.AVAIL' ) );
      with AINDEX^ do
        begin
          NEX := 0; NXI := 0
        end;
      SEEK( AINDEX, 0 ); PUT( AINDEX );
      CLOSE( AINDEX, LOCK ); I := SUCC( I )
    end
  else if LINK <> NULL then
```

```
         begin
           NODE := LINK;
         end
       else begin
              NODE := POINTER[ I ]; I := 1
            end
  end     (* with DICTION^ *)
until NODE = NULL;
PREWORD := ' '; PRECHAP := ' '; PREBOOK := ' ';
REWRITE( PR, CONCAT( PREFIX, 'PRINTER:' ) );
WRITELN( ' USERS PROGRAM.. ' )
```

15

Program Listing for QUERY

15.1 OVERVIEW

QUERY is an example of an extension to the basic MICROBOOK system using the unit stored in SYSTEM.LIBRARY. Although QUERY is not part of the SYSTEM.LIBRARY, its source code is stored on disk volume UNIT:. QUERY uses many of the precompiled routines of unit MICROBOOK.

Most of QUERY consists of routines to scan and interpret QUERY language command statements. Thus, QUERY is a language interpreter which processes very high level language statements. This allows you to process a MICROBOOK database with a minimum of effort. However, the sophisticated user may want to modify QUERY, re-compile, link, and store it on disk volume MICRO: in order to increase its processing capability. Here are some ideas along this line:

(1) Add a command ! which is similar to ?, except it sends the page (and its form) to the printer. This command uses PRINT instead of DISPLAY.

(2) Add a command % which is similar to ?, except it allows you to input a page of values. This command may use INPUT instead of DISPLAY.

(3) Add the ability to format the output in other ways than that defined by a form.

(4) Add the ability to nest FOR-END pairs so that two or more pages can be handled at the same time. This will require a more sophisticated syntax so that you can designate two separate pages.

(5) Add flexibility to the # output page counter so you can specify a page number, etc.

These and other ideas for QUERY can be added to the listing. However, be aware that main memory limitations may limit the practical size of QUERY. A STACK OVERFLOW error will occur if memory is exceeded.

15.2 LISTING

```
(*$S+*)
(*$L PRINTER:*)
program QUERY ;
 uses CHAINSTUFF, MICROBOOK ;
 const
  MAXLINE = 160 ;      (* LIMIT OF INPUT LINE LENGTH *)
 var
  I, OLD, NEWLINE, LINE, PAGENO, OUTPAGE, LMAX : integer ;
  L  : array[ 1..YLIMIT ] of string[ XLIMIT ] ;
  S  : string[ MAXLINE ] ;
  NOFWORD, BOMB, FLAG  : boolean ;
  FWORD, FVALUE, XWORD, FIST, LAST, ALF, PHI, T  : STRWORD ;
  FVAL, FWORDVAL, OLDBETA, BETA, SBETA, GAM : STRWORD ;
  SUM, MIN, MAX, AVG, COUNT, VALD1, VALD2, VALD3, VALT, VALAGG : STRWORD  ;
  D1, D2, D3 : STRWORD;
  CAT, OP, FOP, AOP : char ;
  FFT1, FFT2, FT : real ;
  CLM, LNO: integer ;

 procedure BLANKOUT( var W : STRWORD;
                         C : STRWORD ) ;
  begin
   while POS( C, W ) > 0 do
    begin
     DELETE( W, POS( C, W ), LENGTH( C )  ) ;
    end;
  end ;  (* BLANKOUT *)
```

```
procedure GETLINE ;
 begin
  S := L[ LINE ] ;
  while POS( ' ', S ) > 0 do
   DELETE( S, POS( ' ', S ), 1 );
  while POS( ';', S ) = 0 do
   begin
    LINE := LINE + 1 ;
    if LENGTH( S ) + LENGTH( L[ LINE ] ) > MAXLINE
     then WRITELN( 'INPUT LINE MUST BE LESS THAN ', MAXLINE, ' CHARS' );
    S := CONCAT( S, L[ LINE ] ) ;
    while POS( ' ', S ) > 0 do
       DELETE( S, POS( ' ', S ), 1 ) ;
   end ;
 end ;    (*  GETLINE   *)


procedure ENTER( var LMAX : integer ) ;
 var  N : integer ;
 begin
  PAGE( OUTPUT ) ; GOTOXY( LEFT, UPPER ) ;
  WRITE( 'Enter Query Program : ' ) ;
  LMAX := 0 ;
  repeat
   GOTOXY( XLIMIT+1, YORIGIN+LMAX ) ; WRITE( '!' );
   GOTOXY( XORIGIN, YORIGIN + LMAX ) ;
   LMAX := LMAX + 1 ;
   READLN( L[ LMAX ] ) ;
  until POS( '.', L[ LMAX ] ) = 1 ;
```

```
   repeat
    GOTOXY( LEFT, UPPER ) ;
    WRITE( 'Enter 0(stop, or #(change :' ) ; READLN( N ) ;
    if ( N > 0 ) and ( N < YLIMIT )
     then begin
          GOTOXY( XORIGIN, YORIGIN + N - 1  ) ;
          READLN( L[ N ] ) ;
         end ;
   until N <= 0 ;      (*   NO MORE CHANGES ?       *)

 end ;   (*   ENTER   *)

procedure GETQ ;

 type
  COMMANDS = record
              LL   : array[ 1..YLIMIT ] of string[ XLIMIT ] ;
              CSIZE: integer ;
             end ;                (* command line file   *)
  var
   COMMAND : file of COMMANDS ;

   begin
    PAGE( OUTPUT ) ; GOTOXY( LEFT, UPPER ) ;
    WRITE( 'Enter command program name: ' ); READLN( LINEIN ) ;
    GOTOXY( XORIGIN, YORIGIN ) ;
    RESET( COMMAND, LINEIN ) ;
    with COMMAND^ do
     for NPX := 1 to CSIZE do
      begin
      GOTOXY( XORIGIN, YORIGIN+NPX-1 ) ;
      WRITE( LL[ NPX ] ) ; L[ NPX ] := LL[ NPX ]
      end ;
    LMAX := COMMAND^.CSIZE ;
    CLOSE( COMMAND, LOCK ) ;
   end ;   (*  GETQ   *)

procedure SAVQ ;

 type
  COMMANDS = record
              LL   : array[ 1..YLIMIT ] of string[ XLIMIT ] ;
              CSIZE: integer ;
             end ;            (* command line file   *)
  var
   COMMAND : file of COMMANDS ;

   begin
    PAGE( OUTPUT ) ;   GOTOXY( LEFT, UPPER ) ;
    WRITE( 'Enter program name: ' ); READLN( LINEIN ) ;
    REWRITE( COMMAND, LINEIN ) ;
    with COMMAND^ do
     begin
      CSIZE := LMAX ;
      for NPX := 1 to LMAX do LL[ NPX ] := L[ NPX ] ;
     end ;
    SEEK( COMMAND, 0 ) ;
    PUT( COMMAND ) ;
    CLOSE( COMMAND, LOCK ) ;
   end ;     (* SAVQ   *)
```

```
function SCANOP( P : STRWORD; var Q : STRWORD  ) : boolean;
 begin
  SCANOP := FALSE ;
  if POS( P, S ) > 0
   then begin
    Q := COPY( S, 1, POS( P, S ) + LENGTH( P ) - 1 ) ;
    BLANKOUT( Q, ' ' ) ; BLANKOUT( Q, P ) ;
    DELETE( S, 1, POS( P, S ) + LENGTH( P ) - 1 ) ;
    SCANOP := TRUE ;
   end ;
 end;    (*    SCANOP     *)

function SCANF( P : STRWORD ) : boolean;
 begin
  SCANF := FALSE ;
  if POS( P, S ) > 0
   then begin
    DELETE( S, 1, POS( P, S ) + LENGTH( P ) - 1 ) ;
    SCANF := TRUE ;
   end ;
 end ;      (*    SCANF     *)

procedure SCANW( var Q : STRWORD ; P : STRWORD  ) ;
 begin
  if POS( P, S ) > 0
   then begin
    Q := COPY( S, 1, POS( P, S ) + LENGTH( P ) - 1 ) ;
    DELETE( S, 1, POS( P, S ) + LENGTH( P ) - 1 ) ;
    BLANKOUT( Q, ' ' ) ; BLANKOUT( Q, P ) ;
   end
  else begin
    WRITELN( 'Expecting ', P, ' in ', S ) ;
    WRITELN( 'Strike <rtn>.' ) ;
    READ( KEYIN ) ;
   end ;
 end ;      (*   SCANW    *)

procedure AGG ;

Begin
 if SCANF( 'COUNT' ) then CAT := 'C' else
 if SCANF( 'SUM'   ) then CAT := 'S' else
 if SCANF( 'MAX'   ) then CAT := 'X' else
 if SCANF( 'MIN'   ) then CAT := 'N' else
 if SCANF( 'AVG'   ) then CAT := 'A' else
    begin
     WRITE( 'Error : Expecting aggregate. Strike <rtn>' );
     READ( KEYIN ) ;
     EXIT( AGG ) ;
    end;

(* CAT ALF = BETA TO .. *)

SCANW( ALF, '=' ) ; if POS( '*', ALF ) = 0 then ALF := CONCAT( '*', ALF );

(* BETA TO GAM ..       *)

SCANW( BETA, 'TO' ) ;
SCANW( GAM, 'DO' ) ;
if BETA = XWORD then BETA := FIST ;
if BETA =’*’ then FIRST( ALF, BETA ) ;
if GAM  = XWORD then GAM :=  FIST ;
if GAM  = ’*’ then
 begin
  GAM := ’’ ;
  for I := 1 to MAXWORD do GAM := CONCAT( GAM, ’Z’ ) ;
 end ;

(* PHI OP T            *)

  if SCANOP( ’>=’, PHI ) then OP := ’G’ else
  if SCANOP( ’<=’, PHI ) then OP := ’L’ else
  if SCANOP( ’<>’, PHI ) then OP := ’N’ else
  if SCANOP( ’=’, PHI  ) then OP := ’=’ else
  if SCANOP( ’>’, PHI  ) then OP := ’>’ else
  if SCANOP( ’<’, PHI  ) then OP := ’<’ ;
 (*     T               *)
 T := S ;
 BLANKOUT( T, ’;’ ) ; BLANKOUT( T, ’ ’ ) ;
end;  (*   AGG   *)

procedure AGGREGATE ;
 var PAGE : integer;
     VALUE: STRWORD;
     FT, FVALUE, FMAX, FMIN, FCOUNT, FSUM, FAVG : real ;

 procedure CALC ;
  Begin
   FSUM := FSUM + FVALUE ;
   FCOUNT := FCOUNT + 1 ;
  if FMAX < FVALUE then FMAX := FVALUE ;
  if FMIN > FVALUE then FMIN := FVALUE ;
 end;    (* CALC   *)

begin
 FSUM := 0 ; FCOUNT := 0 ;
 FMAX := 1.2E-37 ; FMIN := 1.7E37 ;
 BOMB := FALSE ;
 SBETA := BETA ;  (*  SAVE BETA FOR SHOWIT  *)
 PAGE := LOOKUP( ALF, BETA ) ;
 repeat
  if PAGE <> NULL then
   begin
    GETWORD( PHI, VALUE, PAGE ) ;
    if not STRFP( VALUE, FVALUE )
      then
       begin
        WRITELN('Numeric error in ', PHI, ' = ', VALUE, ' Strike <rtn>' ) ;
        READ( KEYIN ) ;
       end ;
    if STRFP( T, FT )
     then FPSTR( FT, VALT )
     else
     begin
      GETWORD( T, VALT, PAGE );
      if not STRFP( VALT, FT ) then
        begin
         WRITELN( 'Numeric error in ', T,' = ', VALT,' Strike <rtn>' );
         READ( KEYIN ) ;
        end ;
      end ;
     case OP of
      '=' : if FVALUE = FT then CALC;
      '>' : if FVALUE > FT then CALC;
      '<' : if FVALUE < FT then CALC;
      'G' : if FVALUE >= FT then CALC;
      'L' : if FVALUE <= FT then CALC;
      'N' : if FVALUE <> FT then CALC;
     end ;   (* case *)
    end ;  (* if PAGE <> NULL  *)
   OLDBETA := BETA ;       (*  SAVE FOR SHOWIT  *)
   PAGE := NEXT( ALF, BETA ) ;     (* GET NEXT KEYED VALUE *)
  until ( PAGE = NULL ) or ( BETA > GAM ) or ( BETA = '0' );
  if FCOUNT > 0.0 then begin
    FAVG := FSUM / FCOUNT ;
    FPSTR( FSUM, SUM ) ; FPSTR( FCOUNT, COUNT ); FPSTR( FMAX, MAX ) ;
    FPSTR( FMIN, MIN ) ; FPSTR( FAVG, AVG ) ;
   end
  else BOMB := TRUE ;
end ;     (*  AGGREGATE   *)


function SCANFOR : boolean ;

  var I, J : integer ;
      QUOT : STRWORD ;

  begin
   QUOT := '''' ;
   SCANFOR := FALSE ;
```

```
I := POS( 'FOR', S ) ;
if I > 0
 then  DELETE( S, I, 3 )        (* REMOVE FOR     *)
 else EXIT( SCANFOR );

(* XWORD = FIST TO LAST DO ;                    *)

I := POS( '*', S ) ;   J:= POS( '=', S ) ;
if ( I < J ) and ( I > 0 )
 then XWORD := COPY( S, I, J-I )
 else begin
       WRITELN( 'Error : expecting *INDEX. Strike <rtn>' );
       READ( KEYIN ) ;
       EXIT( SCANFOR )
      end;

BLANKOUT( XWORD, ' ' ) ; DELETE( S, I, J-I+1 ) ;

(*     FIST TO LAST DO ;                        *)

J := POS( 'TO', S ) ;
if J > 0
 then  FIST := COPY( S, 1, J-1 )
 else begin
       WRITELN( 'Error ; expecting TO. Strike <rtn>' );
       read( KEYIN ) ;
       EXIT( SCANFOR )
      end;

BLANKOUT( FIST, ' ' ) ; BLANKOUT( FIST, QUOT ) ;
```

```
if POS( ’*’, FIST ) > 0
 then FIRST( XWORD, FIST ) ;

DELETE( S, 1, J+1 ) ;

(*   LAST DO ;              *)

J := POS( ’DO’, S ) ;
if J > 0
 then LAST := COPY( S, 1, J-1 )
 else begin
       WRITELN( ’Error : expecting DO. Strike <rtn>’ ) ;
       read( KEYIN ) ;
       EXIT( SCANFOR )
      end;

DELETE( S, 1, J+1  ) ;
BLANKOUT( LAST, ’ ’ );  BLANKOUT( LAST, QUOT ) ;

if POS( ’*’, LAST ) > 0
 then
  begin
   DELETE( LAST, 1, LENGTH( LAST ) ) ;
   for I := 1 to MAXWORD do LAST := CONCAT( LAST, ’Z’ ) ;
  end ;

(*  DO FWORD FOP FVALUE;  *)

NOFWORD := FALSE ;
FWORD :=’’ ; FVALUE := ’’ ; FOP := ’ ’ ;
```

```
   if SCANOP( '>=', FWORD ) then FOP := 'G' else
   if SCANOP( '<=', FWORD ) then FOP := 'L' else
   if SCANOP( '<>', FWORD ) then FOP := 'N' else
   if SCANOP( '='  , FWORD ) then FOP := '=' else
   if SCANOP( '>'  , FWORD ) then FOP := '>' else
   if SCANOP( '<'  , FWORD ) then FOP := '<' else NOFWORD := TRUE ;
   BLANKOUT( FWORD, ' ' ) ;
   FVALUE := COPY( S, 1, POS( ';', S ) ) ;
   BLANKOUT( FVALUE, ';' );
   BLANKOUT( FVALUE, ' ' ) ;

   SCANFOR := TRUE ;
end ;  (*    SCANFOR     *)

procedure SHOWIT ;

 begin
  PAGE( OUTPUT ) ; GOTOXY( LEFT, UPPER );
  if POS( '=', S ) = 0
   then
    begin
     if DICLOOK( COPY( XWORD, 1, LENGTH( XWORD ) ),
                            BOOK, CHAPHEAD, CLM, LNO )
         then begin
           DISPLAY( BOOK, CHAPHEAD, CLM, PAGENO );
           GOTOXY( LEFT, UPPER ) ; WRITELN( BLANK ) ;
           GOTOXY( LEFT, UPPER ) ; WRITE( 'BK : ', BOOK, ' CH : ', CHAPHEAD );
           PROMPTS( LEFT, UPPER+1, 'Output to printer (Y/N ) ?', LINEIN );
           if POS( 'Y', LINEIN ) > 0 then PRINT( BOOK, CHAPHEAD, PAGENO ) ;
          end
         else begin
            WRITE( 'No index for key = ', XWORD );
            WRITE( '       Strike <rtn>' );
          end ;
     end  (*   if POS   *)
   else begin
    AGG ; AGGREGATE ;
    GOTOXY( XORIGIN, YORIGIN ) ;
    if not BOMB then
     for LNO := 1 to 5 do
      case CAT of
       'S'  : begin WRITELN( 'SUM   = ', SUM ) ; CAT := 'C' end ;
       'C'  : begin WRITELN( 'COUNT= ', COUNT ) ; CAT := 'X' end ;
       'X'  : begin WRITELN( 'MAX   = ', MAX ) ; CAT := 'N' end ;
       'N'  : begin WRITELN( 'MIN   = ', MIN ) ; CAT := 'A' end ;
       'A'  : begin WRITELN( 'AVG   = ', AVG ) ;CAT := 'S' end ;
      end
    else  WRITELN( 'No match found.' ) ;
   GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER );
   WRITE( ALF,' = ',SBETA,' TO ',OLDBETA,' DO ',PHI,' ',OP,' ',T,' ;' );
   GOTOXY( LEFT, UPPER+1 ) ;WRITE( 'Strike <rtn>.' ) ; READ ( KEYIN ) ;
   end ;   (* else *)
 end ;    (* SHOWIT *)

procedure LET  ;

 procedure AGGEVAL ;
  Begin
   AGGREGATE ;
   if not BOMB then
```

```
    case CAT of
     'S' : VALAGG := SUM ;
     'C' : VALAGG := COUNT ;
     'X' : VALAGG := MAX ;
     'N' : VALAGG := MIN ;
     'A' : VALAGG := AVG ;
    end
   else ZERO( VALAGG ) ;
 end;    (*    AGGEVAL     *)

Procedure AOPEVAL ;
 Begin
  case AOP of
   '+' : ADD( VALD2, VALAGG, VALD1 ) ;
   '-' : SUB( VALD2, VALAGG, VALD1 ) ;
   '*' : MUL( VALD2, VALAGG, VALD1 ) ;
   '/' : DVD( VALD2, VALAGG, VALD1 ) ;
  end ;
 end ;      (*  AOPEVAL    *)

procedure STORED1( VAL : STRWORD ) ;
 Begin
  GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ;
  GOTOXY( LEFT, UPPER ) ;
  if POS( '#', D1 ) > 0
   then
    begin
     BLANKOUT( D1, '#' ) ;
     PUTWORD( D1, VAL, OUTPAGE, POS( '*', D1 ) > 0  ) ;
     WRITE( 'Output ', D1, ' to page = ', OUTPAGE ) ;
     end
    else
     begin
      PUTWORD( D1, VAL, PAGENO, POS( '*', D1 ) > 0  ) ;
      WRITE( 'Output ', D1, ' to page = ', PAGENO )
     end ;
  end ;    (*  STORED1   *)

 procedure OPAGG ;
  begin
   AGG ;
   AGGEVAL ;
   if STRFP( D2, FT ) then FPSTR( FT, VALD2 )
     else GETWORD(  D2, VALD2, PAGENO ) ;
   if not BOMB then AOPEVAL
               else VALD1 := VALD2 ;
  STORED1( VALD1 )
 end;  (*   OPAGG   *)

procedure OPONLY ;
 begin
  SCANW( D3, ';' ) ;
  if STRFP( D2, FT ) then FPSTR( FT, VALD2 )
     else GETWORD( D2, VALD2, PAGENO ) ;
  if STRFP( D3, FT ) then FPSTR( FT, VALAGG )
     else GETWORD( D3, VALAGG, PAGENO ) ;
  AOPEVAL ;
 STORED1( VALD1 )
end ;     (*   oponly    *)
```

```
Procedure AGGONLY ;
 Begin
  AGG ;
  AGGEVAL ;
  if not BOMB
   then STORED1( VALAGG ) ;
 end ;    (*  aggonly   *)

procedure MOVE ;
 begin
  SCANW( D2, ';' ) ;
  VALD2 := D2 ;
  if POS( '''', VALD2 ) >0
    then BLANKOUT( VALD2, '''' )
    else GETWORD( D2, VALD2, PAGENO ) ;
  STORED1( VALD2 ) ;
 end;    (*    move   *)

Begin    (* L E T   *)
 SCANW( D1, '=' ) ;
 AOP := ' ' ;
 CAT := ' ';
 if POS( 'COUNT' ,S ) = 1  then CAT := 'C' else
 if POS( 'SUM'   ,S ) = 1   then CAT := 'S' else
 if POS( 'MAX'   ,S ) = 1   then CAT := 'X' else
 if POS( 'MIN'   ,S ) = 1   then CAT := 'N' else
 if POS( 'AVG'   ,S ) = 1   then CAT := 'A' ;
 if CAT = ' ' then
  begin      (* must not be aggregate, only *)
   if S[ 1 ] = '*' then
```

```
      begin DELETE( S, 1, 1 ) ; CAT := '*' ; end ;
   if SCANOP( '+', D2 ) then AOP := '+' else
   if SCANOP( '-', D2 ) then AOP := '-' else
   if SCANOP( '*', D2 ) then AOP := '*' else
   if SCANOP( '/', D2 ) then AOP := '/' ;
   I := POS( '=', S ) ;
   if CAT = '*' then S := CONCAT( '*', S ) ;
   if ( I > 0 ) and ( AOP <> ' ' ) then OPAGG else
   if ( I = 0 ) and ( AOP <>' '  ) then OPONLY  else
   if ( I = 0 ) and ( AOP = ' ' ) then MOVE ;
  end    (*  all but aggregate, only  *)
 else
  begin  AGGONLY end ;    (* special case  *)
 ZERO( VALD1 ) ; ZERO( VALD2 ) ; ZERO( VALD3 ) ; ZERO( VALAGG ) ;
end ;   (*   LET    *)

procedure PROCESS ;    (* PROCESS A PAGE  *)
 var OK : boolean ;
     NUMB1, NUMB2 : boolean ;   (* numbers or letters ? *)
 Begin
  OK := NOFWORD ;
  if not NOFWORD then
   begin
    GETWORD( FWORD, FWORDVAL, PAGENO );
    NUMB1 := STRFP( FWORDVAL, FFT1 ) ;
    FVAL := FVALUE ; BLANKOUT( FVAL, '''' );
    if not ( POS( '''', FVALUE ) > 0 )
     then
      begin
      GETWORD( FVALUE, FVAL, PAGENO ) ;
  NUMB2 := STRFP( FVAL, FFT2 )
  end
 else NUMB2 := STRFP( FVAL, FFT2 ) ;
OK := FALSE ;
if ( NUMB1 and NUMB2 ) then
   case FOP of
    '=' : OK := FFT1 = FFT2 ;
    '>' : OK := FFT1 > FFT2 ;
    '<' : OK := FFT1 < FFT2 ;
    'G' : OK := FFT1 >=FFT2 ;
    'L' : OK := FFT1 <=FFT2 ;
    'N' : OK := FFT1 <>FFT2 ;
   end
  else
   case FOP of
    '=' : OK := FWORDVAL = FVAL ;
    '>' : OK := FWORDVAL > FVAL ;
    '<' : OK := FWORDVAL < FVAL ;
    'G' : OK := FWORDVAL >=FVAL ;
    'L' : OK := FWORDVAL <=FVAL ;
    'N' : OK := FWORDVAL <>FVAL ;
   end ;
end ;     (* if not NOFWORD      *)

if OK then
 begin
  FLAG := FALSE ;
  OLD := LINE ;                (* SAVE LOOP POINTER   *)
  repeat
   LINE := LINE + 1 ;
```

```
       GETLINE ;
       if POS( '?', S ) > 0 then SHOWIT ;
       if SCANF( 'LET' )  then LET ;
       if POS( 'END', S ) > 0 then
         begin NEWLINE := LINE+1 ; LINE := OLD; FLAG:=TRUE end;
      until FLAG ;  (* FLAG = TRUE MEANS END-OF-FOR *)
      OUTPAGE := OUTPAGE + 1 ;     (* # OUTPUT PAGE NO.   *)
     end ;    (* if OK        *)
end ;    (*  process  *)


  begin
   PAGE( OUTPUT ) ;  GOTOXY( LEFT, UPPER+3 ) ;
   WRITELN( ' Q U E R Y   P R O C E S S O R' );
   WRITELN ; WRITELN ;
   WRITELN('G(et    : get stored commands' );
   WRITELN('S(ave   : save current commands' );
   WRITELN('N(ew    : create new commands' );
   WRITELN('R(un    : run current commands' );
   WRITELN('Q(uit   : stop processor !' );
   repeat
   GOTOXY( LEFT, UPPER ) ; WRITE( BLANK ) ; GOTOXY( LEFT, UPPER );
   WRITE( 'Enter: G(et, S(ave, N(ew, R(un, or Q(uit' );
   READ( KEYIN ) ;
   case KEYIN of
    'G'  : GETQ ;
    'S'  : SAVQ ;
    'N'  : ENTER( LMAX ) ;
    'Q'  : begin SETCHAIN( 'MICRO:MICROBOOK' ) ; EXIT( QUERY ) ; end ;
```

```
end ;
if KEYIN = 'R' then
 begin
  PAGE( OUTPUT ) ;
  LINE := 1 ;
  repeat
   GETLINE ;
   if SCANFOR
    then
     begin
      PAGENO := LOOKUP( XWORD, FIST ) ;
      OUTPAGE := 0 ;   (* OUTPUT PAGE NUMBER  *)
      repeat
       if PAGENO > NULL
        then
         begin
          PROCESS ;
          PAGENO := NEXT( XWORD, FIST ) ;   (* DO IT AGAIN ?     *)
         end ;
      until ( FIST = '0' ) or ( PAGENO = NULL ) or ( FIST > LAST ) ;
      LINE := NEWLINE ;     (* SKIP FOR *)
      end     (*  if SCANFOR   *)
     else
      begin
       XWORD := '' ;
       if POS( '?', S ) > 0 then SHOWIT ;
       LINE := LINE+1 ;
      end ;
    until LINE = LMAX ;
   end ;      (* if KEYIN=R *)
  until FALSE ;
 end.    (* main *)
```

Index

? statement, 74
Accounting system, 13
A(DD BOOK, 21, 29
ADD procedure, 86
A(DD TO FILE, 45
Aggregate expression, 73
Aggregates, 73
AVG, 73
B-tree, 2
Billing system, 13
Blank entry, 33
Blank space, 38
B(ROWSE, 21
Browse errors, 53
Browse prompts, 52
Browsing, 51
CHAINSTUFF, 85
Chapter, 16
Columns, 35
Command sequence, 22
Command summary, 19
COMP:, 17
CONVERT, 27
CONVERT routine, 106
COP(Y, 21, 79
COP(Y command, 79
COUNT, 73
C(REATE, 21
C(REATE purpose, 23
Data dictionary, 40
Data structure listing, 113
Data structure of
  MICROBOOK, 86, 104
Data value, 9, 33, 47
Data value display, 93
Data word, 9, 33
Data word display, 92
DBMS, 1
Delete copy, 81
DICLOOK function, 87
Disk, 15
DISPLAY procedure, 88
Domain, 5
DVD procedure, 86
EMPLOYEE, 10
END QUERY, 70
END statement, 71
E(NTER PAGES, 21, 45
Example of accounts
  receivable, 91
Example of Pascal
  extension, 89
EX(ECUTE command, 65
Execution errors, 84

Extending books, 83
Extensions, 85
FIRST procedure, 87
FOR QUERY, 70
FOR statement, 71
Form, 16
Form input, 33
Form rules, 33
F(ORMS INPUT, 21
G(ET QUERY, 65
GETPAGE procedure, 88
GETWORD procedure, 87
Graphics system, 13
Hardware and software
  requirements, 15
Hashing, 2
Heirarchical, 4
Index, 11
Input/output routines, 88
INPUT procedure, 88
Key word, 11, 34
LET QUERY, 70
LET statement, 72
Library, 42
Limits, 23, 27, 35, 81
Limits, warning, 84
Logical model, 1, 3
LOOKUP function, 87
MASTER file, 13, 91
MAX, 73
Memory space, 27
MICRO:, 15
MICROBOOK definition, 9
MICROBOOK extensions, 85
MICROBOOK listing, 109
MICROBOOK routines, 86
MICROBOOK size, 83
MICROBOOK use, 12
MIN, 73
MODIFY, 27
MODIFY routine, 106
MUL procedure, 86
Network, 4
N(EW FILE, 45
N(EW QUERY, 65
NEXT function, 87
P-code, 16
Page extended, 83
Pascal routines, 86
Password protection, 29
PAYROLL, 10
Physical model, 1
POST QUERY, 97
Power-on, 18
Pre-sort, 45
PRINT procedure, 88, 236
PRINT routine, 236
Printer, 236
Program listing, 103
PUTPAGE procedure, 88
PUTWORD procedure, 87
QUERY.Code, 17
QUERY examples, 76
QUERY language, 5, 65, 69
QUERY processing a/r, 97
QUERY statements,
  summary, 75
Question mark, 70
Question statement, 74
Q(UIT QUERY, 67
Real-estate database, 13
Relational, 4
Reset, 18
Retrieval procedures, 87
Rules of form input, 33
R(UN QUERY, 67
S(AVE COP(Y, 81
S(AVE QUERY, 65
SETCHAIN, 85
Size of book, 83
Sort, 45
SOURCE:, 107

Stack overflow, 27
Startup, 17, 18
Statement summary, 75
Statements file, 91
S(TOP, 21
String overflow, 23, 26, 29, 30
SUB procedure, 86
SUM, 73
System.Library, 17
Transaction file, 91
Tuple, 5
UNIT:, 85, 108
UNIT: listing, 235
Uses, 86
Value range error, 23, 26,
  29, 30

About the Author

Ted Lewis is a veteran computer book author with countless papers/magazine articles and ten previous books to his credit. Listed in the *International Who's Who in Education*, he is currently an Associate Professor of Computer Science at Oregon State University where he is involved in a number of research projects.

MORE HELPFUL WORDS FOR YOU

Computers for Everybody

Jerry Willis and Merl Miller

This fun-to-read book covers all the things you should know about computers. If you're anxious to buy one, use one or just want to find out about them, read this book first.

ISBN 0-918398-49-5 $5.95

Peanut Butter and Jelly Guide to Computers

Jerry Willis

This entertaining book is a simple, easy-to-digest source of information on personal computing. It leads you through all the essential knowledge of "computer literacy."

ISBN 0-918398-13-4 $9.95

Nailing Jelly to a Tree

Jerry Willis and William Danley, Jr.

This is a book about software. The emphasis is on learning to use the thousands of available programs that have already been written, and adapting them to your machine.

ISBN 0-918398-42-8 $15.95

Small Computers for the Small Businessman

Nicholas Rosa and Sharon Rosa

If you've ever considered a computer for your business but didn't know where to turn, this is the book that will arm you with all the information you'll need to make an intelligent, cost-effective decision.

ISBN 0-918398-31-2 $16.95

dP **dilithium Press, P.O. Box 606, Beaverton, OR 97075**

Send to: dilithium Press, P.O. Box 606, Beaverton, OR 97075

Please send me the book(s) I have checked. I understand that if I'm not fully satisfied, I can return the book(s) within 10 days for full and prompt refund.

____ Computers for Everybody
____ Nailing Jelly to a Tree
____ Peanut Butter and Jelly Guide to Computers
____ Small Computers for the Small Businessman

☐ Check enclosed $________
Payable to dilithium Press
☐ Send me your catalog of books.

☐ Please charge my
☐ VISA ☐ Mastercharge
____________ Exp. Date ________
Signature ____________________

Name ____________________
Address ____________________
City, State, Zip ____________________ M

MICROBOOK Software
★★★Order Form★★★

dilithium Press is pleased to offer you the programs from this book on diskettes. The following formats are available:

MICRO: This is a single diskette containing the executable object code of MICROBOOK. CODE, SYSTEM LIBRARY, AND QUERY. CODE.

SOURCE:, UNIT: This is a diskette pair containing all course programs written in Pascal.

☐ Please send me the following diskette(s):
________ MICRO: Object code $15.95 order no. 00030
________ SOURCE:, UNIT: Pascal Source $49.95 order no. 00031

☐ Enclosed please find my check or money order which includes $1 for shipping and handling. $______________.

☐ Please charge my ______Visa ______ Mastercharge

Acct. #________________________ Exp. Date________________________

Signature ____
(or call our toll free number 800-547-1842 to expedite your charge order)

☐ Please send me your catalog of books and software

NAME ____

ADDRESS ____

CITY, STATE, ZIP ____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 143 BEAVERTON, OREGON

Postage will be paid by addressee

dilithium Software

P.O. Box 606
Beaverton, OR 97075

MICROBOOK

At last, here is an affordable way for you to have a database management system on your Apple II! These programs turn your Apple II into a combination filing cabinet, information gathering/retrieval system and data processing engine.

Written in Pascal, the programs simulate a library. Information is maintained and broken down categorically into books, chapters, pages and index to pages—all on diskette. These can be accessed with key words, so the only thing that is necessary on your part is a rational approach in entering the data. Photographs of the Apple II screen are abundant and they show you step-by-step the effect of each of your entries.

MICROBOOK can be used for almost any application involving the storage and retrieval of information—real estate records, stock market histories, family trees, educational testing and quizzes—the possibilities are limitless!

ISBN 0-88056-072-X

Lewis

MICROBOOK: DATABASE MANAGEMENT FOR THE APPLE II®

0-88056-072-X